



In-Train Positioning System

Master Thesis



Public compilation

Abstract Design and implement a multi-input positioning system for use in trains.

ID MAS-06-02.23

Keywords GPS, WiFi, INS, SQL, WCF, train, C#, .NET

Author Marcel Suter
Kanalweg 3, CH-3125 Toffen
+41-(0)76-387 70 06
marcel.suter@gmx.net

Supervisor Stefan Bigler, ENKOM Inventis AG
+41-(0)31-950 42 42
stefan.bigler@enkom.com

Expert Rolf Wenger, Infobrain AG
Rolf.Wenger@infobrain.com

Class MAS-IT 2006-02

Date 10. Oct. 2008

-
- A) Feasibility Study
 - B) SRS
 - C) Main Documentation
 - D) Quick Start Guide
 - E) Time planning



In-Train Positioning System

MAS-06-02.23



Feasibility Study (public version)

Abstract

Design and implement a multi-input positioning system for use in trains.

Author

Marcel Suter
Kanalweg 3
CH-3125 Toffen
marcel.suter@gmx.net

Expert

Rolf Wenger
Infobrain AG
Rolf.Wenger@infobrain.com

Class

Supervisor

Stefan Bigler
ENKOM Inventis AG
stefan.bigler@enkom.com

Date

MAS-IT 2006-02

10. Oct. 2008

Revision History

Tag	Date	Author
First draft	7.10.2008	Marcel Suter
Additions	21.10.2008	Marcel Suter
Public version: Inertial Prices removed WiFi Code removed	3.11.2008	Marcel Suter

Table of Contents

1. Management Summary.....	4
2. Introduction.....	5
3. On inertial navigation.....	6
3.1. Preface.....	6
3.2. Requirements for a suitable inertial navigation device.....	6
3.3. Commercially available devices (prices removed).....	6
3.4. Conclusion.....	13
4. On using WiFi.....	14
4.1. Preface.....	14
4.2. War driving with a laptop in the train.....	14
4.2.1. Bern S-Bahn.....	14
4.2.2. Zurich S-Bahn.....	14
4.2.3. Intercity Zurich-Bern.....	14
4.3. War driving with a mobile device in the train.....	15
4.4. Conclusion.....	15
5. On code for WiFi scanning.....	16
5.1. Preface.....	16
5.2. Existing Projects.....	16
5.3. Own Code in C#.....	16
5.4. Managed WiFi API.....	16
5.5. Advanced WiFi Manager.....	16
5.6. Existing Code from Enkom Inventis (removed).....	17
5.7. Conclusion.....	17
6. On using GPS.....	18
6.1. Preface.....	18
6.2. Measurements.....	18
6.3. Evaluation of Hardware.....	18
6.3.1. UBLOX ANTARIS GPS Positioning Engine.....	18
6.3.2. NaviLock	19
6.3.3. ublox EVK-5H Evaluation Kit with KickStart.....	19
6.4. Conclusion.....	19
7. On using the track database.....	20
7.1. Preface.....	20
7.2. Analysation.....	20
7.3. Conclusion.....	20
8. Bibliography.....	21

1. Management Summary

On the base of the subject proposal, for the master thesis "In-Train Positioning System" [SUM08TE], a feasibility study was carried out. This document describes the study and its conclusions.

The goal of the feasibility study was to determine, which intended technologies were usable for incorporating into the positioning system, and to order the corresponding hardware.

2. Introduction

Pursuant the subject proposal in [SUM08TE] this feasibility study evaluated four technologies:

- inertial navigation
- GPS
- WiFi
- Use of a track database

For the WiFi part, also suitable code for acquiring the required data was evaluated.

3. On inertial navigation

3.1. Preface

The currently used inertial navigation solution is imprecise and should be replaced. The goal is to find a suitable hardware that provides reliable positioning in the 3D space. On inertial navigation systems, a variety of articles are available.

On Wikipedia: http://en.wikipedia.org/wiki/Inertial_navigation

General Description of inertial navigation, its accuracy and used technologies:
<http://www.media.mit.edu/resenv/classes/MAS836/Inertialnotes/DraperOverview.pdf>

Students at the ETH Zurich did build a "Low Cost Inertial Navigation System" as a project. Description is here <http://www.trash.net/~luethi/study/ins/ins.html>. The project concludes, that accurate positioning data though low cost sensors deteriorates very fast.

Kevin J. Walchko of the University of Florida has carried out a project called "Embedded Low Cost Inertial Navigation System". <http://www.mil.ufl.edu/publications/fcrar03/Walchko-2.pdf>. It shows how inertial navigation can be combined with a GPS receiver and how Kalman Filtering is applied therefore.

A. D. KING, B.Sc., F.R.I.N., of Marconi Electronic Systems Ltd. Provides an overview of Inertial Navigation here: http://www.imar-navigation.de/download/inertial_navigation_introduction.pdf

On Evaluating the properties of an inertial navigation system, this article by imar may help:
http://www.imar-navigation.de/beispiele/decision_assistant.pdf

3.2. Requirements for a suitable inertial navigation device

The ideal inertial navigation device for use in this thesis should have the following features:

- USB-port for fast data transfer and power supply
- software library for easy access in .NET
- Ability to indicate a start position and regular updates via the NMEA protocol.
- Built-In Kalman filter
- probably an internal GPS
- accurate also on high speeds
- full 3D positioning using 3 accelerometers and 3 gyros
- Not depend on an odometer, since we do not have access to odometric data in the train cars.

3.3. Commercially available devices (prices removed)

There are several devices on the market, as follows:

On inertial navigation

On inertial navigation

Company/Product	Link	Sample Picture	features	Comment
Honeywell, DRM™4000 Dead Reckoning Module	<p>http://www.magneticensors.com/datasheets/DRM4000.pdf</p> <p>Ordering No: DRM 4000</p> <p>Overview of similar products: http://www.magneticensors.com/products.html</p> <p>Contact Customer Service at 1-800-323-8295 for more information.</p>		<p>Blends GPS and dead reckoning data (via internal sensors) to an NMEA output.</p> <p>Suited for personnel on foot.</p>	<p>This just a pedometer, as clarified by a technician from Honeywell and it will not work in trains.</p> <p>DRM 5 and GyroDRM suffer from the same issue, I was told.</p>
Honeywell DRM® - 5 Dead Reckoning Module, Evaluation Kit Available	<p>http://www.ssec.honeywell.com/magnetic/datasheets/drm5.pdf</p>		<p>Similar to DRM 4000, with built-in GPS receiver.</p> <p>Evaluation Kit available. No USB, powered by an internal rechargeable Battery.</p> <p>By specification, this is not to be used on vehicles.</p> <p>Export-Controlled by ITAR.</p>	<p>Since no external GPS is attachable, initializing the device seems impossible.</p> <p>Not usable in trains.</p>

On inertial navigation

Honeywell GyroDRM	http://www.ssec.honeywell.com/magnetic/datasheets/gyrodrm.pdf		Gyro stabilized inertial navigation for personnel on foot. Has a built-in GPS receiver. A Evaluation Kit is available.	Seems to be an older version of DRM 5. Not usable in trains.
IMAR iDRPOS:	http://www.imar-navigation.de/datenbl/drp os_2pages_e.pdf		2D Navigation module using an external GPS receiver, attachable via RS-232/NMEA. An option for 3D is available.	Is not 3D. According to a sales person, this solution is not eligible for use in trains without odometer, because it would deteriorate fast.
Applanix POS TG	http://www.applanix.com/products/postg_index.php		Inertial, optical and GPS sensors provide millimeter-accuracy on tracks.	Large, and much too precise (probably also too costly)
Gladiator Technologies, Inc. LANDMARK20 AHRS DEMO KIT	http://www.gladiatortechnologies.com/DATASHEET/LandMark20_AHRS_DEMO_KIT_datasheet_051508.pdf		Inertial sensors in one housing, with USB connection. These sensors are mainly developed for sailing and flight application. A Evaluation Kit is available. Integrating and combination of the inertial data and external GPS fixes would have to be done in external software.	This is not a turn-key solution for this thesis, since integration of the inertial data is not done yet. Because integrating is not provided, the precision is probably questionable.

On inertial navigation

OXTS Inertial+	http://www.oxts.co.uk/default.asp?pageRef=102	 A red rectangular electronic device with various ports and connectors on the front panel. The word "Inertial" is printed on the top right corner.		<p>Augments the NMEA data of an external GPS receiver with inertial measurements.</p> <p>There are also other models available.</p> <p>Drift without odometer is about 50m/minute</p>	This product would be quite fine, but I don't have the budget for it.
VibTel RT3000	http://www.vibtel.com/EN/Products/InertialNavigationSystems/StrapdownInertialNavigationSystems/OXTS.MODEL.RT-3000.html	 A red rectangular electronic device with a small circular dial or switch on the front panel. The word "RT3000" is partially visible on the side.		<p>Supports an external GPS and provides data via CAN, RS-232, Ethernet.</p> <p>VibTel seems to be a reseller of OXTS.</p>	Way beyond budget.

On inertial navigation

Sirf SiRFDiRect	http://www.sirf.com/products/SiRFDiRect_Product_Insert.pdf	 	<p>Enhanced GPS + Dead Reckoning Software for Portable Devices.</p> <p>It sports 3 accelerometers and 1 gyro.</p> <p>There is an evaluation Kit available, that provides USB and RS-232 connection. Supply voltage is 9-24 Volts.</p> <p>NMEA protocol is supported.</p> <p>Dead Reckoning precision after 1 Minute is 225 meters.</p> <p>No direct access to the sensor data is possible.</p>	<p>Good option, similar to the MIT-G, apart that we will not have access to sensor data for our own processing.</p>
NovaTel SPAN-CPT	http://www.novatel.com/products/span_cpt.htm		<p>Combines GPS and inertial navigation. GPS is built in the device.</p>	<p>No possibility to feed our own GPS signal into the unit.</p> <p>Too costly.</p>
Xsens MT9			<p>This seems to be an earlier version of the MTx.</p> <p>This product is discontinued.</p>	<p>No possibility for attaching a GPS directly. Mathematical blending of GPS and inertial data would have to be done programmatically.</p>

On inertial navigation

Xsens Mtx	http://www.xsens.com/en/products/human_motion/mtx.php		<p>The MTx is a small and accurate 3DOF Orientation Tracker. It provides drift-free 3D orientation as well as kinematic data: 3D acceleration, 3D rate of turn and 3D earth-magnetic field.</p> <p>No external GPS. Evaluation Kit available.</p> <p>Noise of the accelerometers: $0.001\text{m/s}^2/\sqrt{\text{Hz}}$</p>	No GPS, therefore no absolute referencing possible.
--------------	---	--	---	---

On inertial navigation

Xsens MTi-G	http://www.xsens.com/en/products/machine_motion/mtig.php	 The image shows two items: a red rectangular Inertial Measurement Unit (IMU) at the top, and below it, an open black carrying case containing a smaller orange component and a coiled black cable.	<p>The MTi-G is a MEMS based Inertial Measurement Unit (IMU) and has an onboard Attitude and Heading Reference System (AHRS) combined with GPS and a static pressure sensor.</p> <p>There is a Sensor Development Kit available which contains the sensor and some accessories plus the SDK. The SDK has various possibilities to access the data, namely a WIN32 DLL.</p> <p>Deterioration according to a technical seller is 350m in a minute.</p> <p>The GPS has a tracking sensitivity of -158dBm according to specification.</p> <p>Does work without GPS for only 10 seconds. After that, only orientation information is provided.</p> <p>Noise of the accelerometers: $0.002\text{m/s}^2/\sqrt{\text{Hz}}$</p>	No possibility for an external GPS. We can not update the reference position.
----------------	---	---	---	---

On inertial navigation

Xsens MTI	http://www.xsens.com/en/products/machine_motion/mti.php		<p>The MTi is a miniature, gyro-enhanced Attitude and Heading Reference System (AHRS). Its internal low-power signal processor provides drift-free 3D orientation as well as calibrated 3D acceleration, 3D rate of turn and 3D earth-magnetic field data.</p> <p>No GPS attachable.</p> <p>Noise of the accelerometers: $0.002\text{m/s}^2/\text{sqrt(Hz)}$</p>	<p>Just data available. No position data is calculated.</p> <p>This is thus similar to our available sensor, the MT9.</p>
--------------	---	--	---	---

3.4. Conclusion

From a technical an implementation's point of view, the Inertial+ device is the most suitable, because it is the most accurate one, and provides custom input of pre-calculated position estimates with advanced kalman filtering already solved.

From a product point of view, one of the Xsens products without GPS makes most sense, because they are cheaper and smaller. However, in the thesis, there will be not enough time to solve the advanced mathematics involved with this.

SirfDiRect would also be a solution for the thesis. However, for a later product, this does not provide the needed flexibility, since no access to the sensor data is possible.

As a compromise, I recommend the Xsens MTI-G. It provides both the ease-of-use that is required for easy integration in the project, and the flexibility to later add customized advanced mathematical algorithms.

4. On using WiFi

4.1. Preface

The goal is to find out, how good the reception of WLAN Access Points in trains is. Therefore I used two systems trying to scan for surrounding Access points. One was a standard laptop, and one a mobile phone.

4.2. War driving with a laptop in the train

A Thinkpad X60 with a built-in WiFi Antenna plus an external GPS (for reference only) was used. The actual scanning was done using the Netstumbler software.

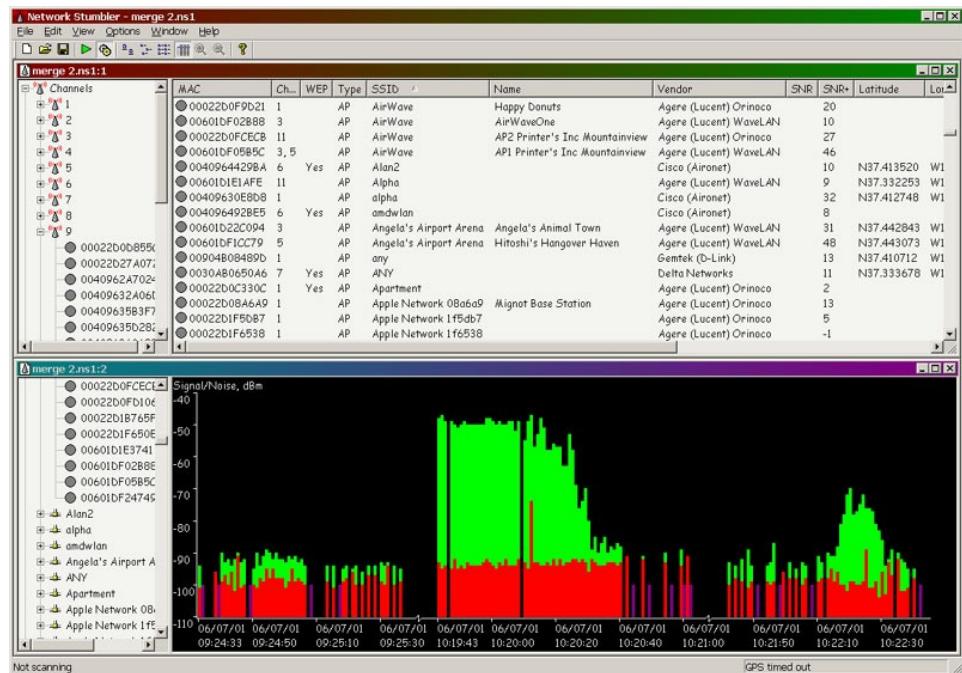


Illustration 1: Netstumbler in Action

4.2.1. Bern S-Bahn

Measurements on these (older) trains showed many access points not only in stations but also while moving.

4.2.2. Zurich S-Bahn

In the cars of the Zurich S-Bahn, the WLAN reception was good, continuously some Access Points were showing up, usually between 2 and 20.

4.2.3. Intercity Zurich-Bern

Near stations and when driving through populated areas there were always more than one Access Point available. In open areas, only rarely an Access Point is visible.

Swisscom operates public non-free Access Points in many larger stations, including Bern, Aarau and Zurich. they are receivable from within the train, even when driving trough a station at high speed, eg. in Aarau.

The cars have their own Access Points to provide the passengers with payable Internet access. These Access Points must get marked and stored as non-useable for positioning purposes. There should also be a general plausibility check for positions when doing positioning with Access Points as they may change position at will. Examples may be if a person owning a private Access Point moves or when the Access Point is in another train car.

In tunnels, no Access Points were detected, except the ones from the car itself.

4.3. War driving with a mobile device in the train

A commercially available mobile phone, the Trinity 100 from HTC was used. It has a built-in WiFi and also built-in GPS. The WiFiFoFum software was used for sniffing.

In urban areas, many Access points were picked up, sometimes up to 16 at the same time. When traveling faster, especially in the Intercity, few, but still some, Access points were picked up.

4.4. Conclusion

Using WiFi access points for positioning purposes in trains is very feasible. The results are expected to get even better if an external antenna would be used, instead of built-in ones.

Especially WiFi is suitable to replace the manual markers of the current solution, because at or near train stations there are generally many Access Points receivable.



Illustration 2: Seeing 7 Satellites and two active Access points

5. On code for WiFi scanning

5.1. Preface

To get information about a laptop's surrounding WiFi networks, software is necessary. This software could either be written in the master thesis, or an existing library could be bought. A Linksys Wireless-G Adapter „WUSB54GC“ was used for evaluating. The important part of the information is the SSID and the MAC Address of a network.

I have studied several methods to reliably get data from surrounding networks.



Illustration 3: Wireless USB Adapter from Linksys

5.2. Existing Projects

There are already several projects and solutions in this field, including:

Placelab

<http://www.deviceforge.com/articles/AT8606455669.html>

It consist of Open Source software and multiple databases, including self-created ones, could be used. There is a complete paper explaining the concept and the architecture of the software in [PLLB01].

SkyHook

<http://www.skyhookwireless.com/howitworks/>

Navizon

<http://www.navizon.com/>

Ekahau

<http://ekahau.com/?id=4200>

5.3. Own Code in C#

A first try with writing own code was unsuccessful. (See the Solution "WifiDetector" in the code directory).

5.4. Managed WiFi API

I have found an open source project, Managed WiFi API, which works with Windows XP SP2, but is not capable to retrieve the MAC address of surrounding networks [CPLX01].

5.5. Advanced WiFi Manager

Advanced WiFi-Manager [NICO01], is capable of getting both the SSID and the MAC, including the signal strength of course. It provides a sample program in C#, showing it's capabilities.

This library would suit our needs.

5.6. Existing Code from Enkom Inventis (removed)

5.7. Conclusion

Multiple solutions are possible. I will evaluate again and decide on one after the SRS is written, because the using of the WiFi technology might be optional.

6. On using GPS

6.1. Preface

I have tested the GPS reception in trains using a NaviLock GPS receiver with a high sensitivity of -160dB.

6.2. Measurements

The measurements were done using the same Hardware as for the WiFi test drive.

Zurich S-Bahn

In the car of the Zurich S-Bahn I experienced very good GPS reception, with 7 to 8 satellites used. The speed of the train was around 70 km/h max.



Illustration 4: Test drive for GPS usability in trains

Intercity Zurich-Bern

In the car of the Intercity from Zurich to Bern the reception of the GPS was varying. In the Zurich Main Station there were 5 to 6 satellites used. While moving slowly and the sky was open, up to 8 satellites were used. At traveling with about 100 km/h and open sky, 3 to 5 Satellites were used. When the train moves slower, reception is generally better.

In tunnels, of course, no fix was available. When exiting a tunnel, it lasts about 30 seconds before the first fix again.

After a forced cold start in the moving train,(at around 100 km/h in the open field) the receiver did not recover for the rest of the drive (at least 30 minutes).

6.3. Evaluation of Hardware

Several GPS receivers are available on the market. I have evaluated some.

6.3.1. UBLOX ANTARIS GPS Positioning Engine

This is a stand-alone GPS receiver, build as an evaluation Kit with external Antenna and an RS 232-Output. Power comes from an external 9VDC Adaptor.

This device is a little outdated but is already available in the company. It's disadvantage however is that it does not feature an USB port.

6.3.2. NaviLock

The company has a GPS receiver from Navilock which has a specified input sensitivity of -160dBm. It was used to make the feasibility test drive in the train.

6.3.3. ublox EVK-5H Evaluation Kit with KickStart

The EVK-5H is an evaluation kit of a GPS chip with the so-called "KickStart" and "SuperSense" technology. This provides fast TFF and accurate results with low reception levels. Information is provided here:

http://www.u-blox.de/products/evk_5h.html

Tracking sensitivity is -160 dBm.

Price is EUR 150.00 (Exclusive V.A.T.)



Illustration 5: EVK-5H Evaluation Kit

6.4. Conclusion

The usage of GPS is feasible. To have good results, the GPS receiver should have already a fix when boarding the train.

I ordered the EVK-5H Evaluation Kit from uBlox.

7. On using the track database

7.1. Preface

The information of all relevant tracks of the swiss federal railways is available as raw, comma separated coordinate values in the swiss coordinate system, in plain text files. They are each named after the logical track number according to the numbering scheme of the swiss federal railways.

7.2. Analysation

The accuracy of these coordinate is quite good, often around several meters. An analzyation of the data in Google Earth shows this, see below. Tracks are shown as white lines.

7.3. Conclusion

The usage of this data is feasible. I will have to build a custom database to hold that data, and import it manually. To keep the database design and administration simple, there will be most likely no automatic update of tracks with either new text files or on-the-fly from acquired position fixes during drives.

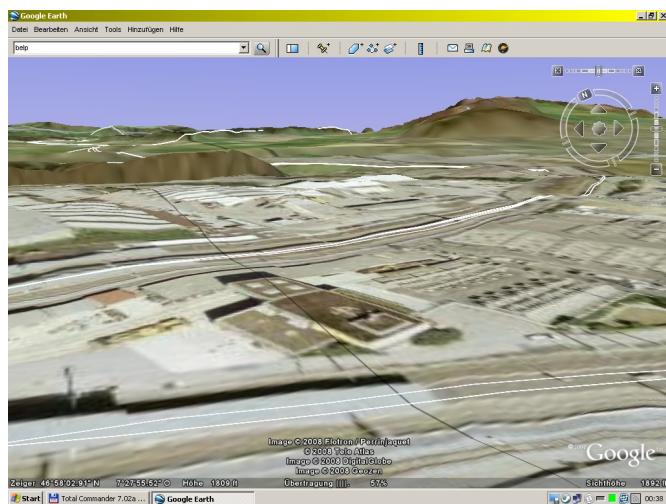


Illustration 6: Available track data of swiss railways, detail of the Wankdorf station, near Bern

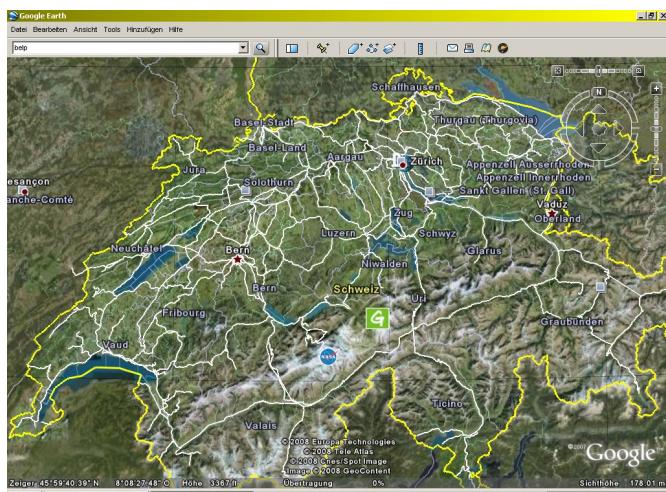


Illustration 7: Available Track data of swiss railways, overview

8. Bibliography

[SUM08TE]	Themeneingabe Master Thesis Thema In-Train-Navigation, 21.8.2008, Marcel Suter. Obtainable from the author.
-----------	--

In-Train Positioning System

MAS-06-02.23



SRS

Abstract	Design and implement a multi-input positioning system for use in trains.
Keywords	GPS, WiFi, INS, SQL, train, C#, .NET
Author	Marcel Suter Kanalweg 3, CH-3125 Toffen +41-(0)76-387 70 06 marcel.suter@gmx.net
Supervisor	Stefan Bigler, ENKOM Inventis AG +41-(0)31-950 42 42 stefan.bigler@enkom.com
Expert	Rolf Wenger, Infobrain AG Rolf.Wenger@infobrain.com
Class	MAS-IT 2006-02
Date	10. Oct. 2008

Revision History

Tag	Date	Author
Draft	21.10.2008	Marcel Suter
Small changes and additions	22.10.2008	Marcel Suter
CR's from the Supervisor	24.10.2008	Marcel Suter
Small add-ons about IC2000	27.10.2008	Marcel Suter
Assumed Availability of Hardware added	10.11.2008	Marcel Suter
CR's from the Expert	11.11.2008	Marcel Suter

Table of Contents

1. Introduction.....	4
1.1. Purpose.....	4
1.2. Scope.....	4
1.3. Definitions, acronyms, and abbreviations.....	4
1.3.1. Definitions.....	4
1.3.2. Acronyms and Abbreviations.....	5
1.4. References (Bibliography).....	7
1.5. Overview.....	8
2. Overall description.....	9
2.1. Product perspective.....	9
2.1.1. System interfaces.....	9
2.1.2. User interfaces.....	9
2.1.3. Hardware interfaces.....	9
2.1.4. Software interfaces.....	9
2.1.5. Memory.....	10
2.1.6. Operations.....	10
2.1.7. Site adaptation requirements.....	10
2.2. Product functions.....	10
2.3. User characteristics.....	11
2.4. Constraints.....	12
2.5. Assumptions and dependencies.....	12
2.6. Apportioning of requirements (2.6 of the SRS).....	13
3. Specific requirements.....	14
3.1. External interface requirements.....	14
3.1.1. System Interfaces.....	14
3.1.2. User Interfaces.....	15
3.1.3. Hardware Interfaces.....	17
3.1.4. Operations.....	17
3.2. Functions.....	17
3.2.1. Using WiFi as a position provider.....	17
3.2.2. Using GPS as a position provider.....	18
3.2.3. Using an IMS as position provider.....	18
3.2.4. Using a track database for correction.....	18
3.2.5. Using event markers as position provider.....	18
3.2.6. Logging.....	18
3.2.7. Estimating the position.....	19
3.3. Performance Requirements.....	19
3.4. Database requirements.....	19
3.5. Design constraints.....	19
3.5.1. Standards compliance.....	19
3.6. Software system attributes.....	20
3.6.1. Source Code.....	20
3.6.2. Availability.....	20
3.7. Other requirements.....	20
Appendix A: Overview of the existing system.....	21

1. Introduction

1.1. Purpose

This Software Requirement Specification (SRS) describes requirements to the system that is to be developed in the Master Thesis "In-Train Positioning System", Version 1.0. From this SRS, a time schedule is derived for the thesis.

The intended audience for this SRS, besides the student, are all the supervisor, the expert, and the customer for this thesis plus anyone interested in the topic. A basic knowledge of navigation and computer systems is assumed.

The structure of this SRS is derived from the IEEE Standard 830-1998, described in [IEEE98]

1.2. Scope

The system provides position estimates in moving trains. It will use various technologies for deriving positioning information. The main benefit of this system is to replace an older one, that only produced inertial and gyro data in a raw format, instead of calculating the current position.

The system consists of a portable computer, an executable software component, and the necessary hardware. The system will be portable by a single person and be mountable on a trolley.

1.3. Definitions, acronyms, and abbreviations

1.3.1. Definitions

Term	Description	Hyperlink(s)
Engine executable	An executable, that contains the core functionality of the system. It may run independently from other components.	
CBPE	Current best position estimate	
Current best position estimate	A mixed position estimate, that combines information from the various position providers in the system. This estimate is extrapolated, if needed to provide an up-to-date position of the current moment in time.	
Event marker	This denotes a defined landmark, with a known geographical location. In the existing system, event markers are fired by the user when traveling past the	

Introduction

	<p>landmark and stored in the measurement files. They are used for the positioning in the post-processing step.</p> <p>In the new system they are still used, but directly transformed into a positioning information.</p>	
Event marker database	A local database where all event markers are stored.	
Measurement system	<p>This is used to denote any attached device to the system, that uses the position estimates provided by this system.</p> <p>The attached device may be of any kind. However, in the currently targeted environment of this system, this will be most likely any form of radio frequency measurement system.</p>	
Position provider	A sensor, probably with additional preprocessing, or some other input that provides positioning information to the system.	
Railway segment	English term for the "Fahrplanfeld" in german. This denotes a logical connection between two nodes. In Switzerland all railway tracks are part of such a "Fahrplanfeld".	
Railway segment number	For public transportation trains in Switzerland, railway segments are usually numbered using three digits. For example the connection Bern-Olten has the number 450.	http://de.wikipedia.org/wiki/Liste_von_Eisenbahnstrecken_in_der_Schweiz , in German.

1.3.2. Acronyms and Abbreviations

Acronym / Abbreviation	Description	Hyperlink(s)
AHRS	Attitude Heading Reference System	http://en.wikipedia.org/wiki/Attitude_and_Heading_Reference_Systems
API	Application Programming Interface	
CEP	Circular error probable	http://en.wikipedia.org/wiki/Circular_error_probable
DOF	Degrees of freedom. In this thesis this is often used as 6DOF. This usually means, that the referred component uses 3 gyros and 3 accelerometers.	http://en.wikipedia.org/wiki/Degrees_of_freedom_(mechanics)
GGA	Stands for "Global Positioning System Fix Data", a part of the NMEA protocol which contains essential fix data which provide 3D location and accuracy.	http://www.gpsinformation.org/dale/nmea.htm#GGA
GPS	Global Positioning System. The Global Positioning System (GPS) is the only fully functional Global Navigation Satellite System (GNSS) in the world. It was developed by the United States Department of Defense.	http://en.wikipedia.org/wiki/GPS
INS	Inertial Navigation System	http://en.wikipedia.org/wiki/Inertial_navigation_system
ITAR	International Traffic in Arms Regulations (ITAR) is a set of United States government regulations that control the export and import of defense-related articles and services on the United States Munitions List.	http://en.wikipedia.org/wiki/ITAR
MAC	Media Access Control, a quasi-unique identifier	http://en.wikipedia.org/wiki/MAC_Address
MEMS	Microelectromechanical systems	http://en.wikipedia.org/wiki/MEMS
NMEA	National Marine Electronics Association. The term NMEA most often refers the protocol called "NMEA 0183", issued by the organization, a defacto standard for providing GPS-related positioning information.	http://www.nmea.org/ http://en.wikipedia.org/wiki/NMEA

Introduction

NMEA Sentence	A part of information described by the NMEA Protocol.	http://en.wikipedia.org/wiki/NMEA
RAM	Random Access Memory	http://en.wikipedia.org/wiki/Random_access_memory
RMC	"Recommended Minimum sentence C", a part of the NMEA protocol	http://www.gpsinformation.org/dale/nmea.htm#RMC
RS-232	Widely used protocol and interface specification for binary data transfer	http://en.wikipedia.org/wiki/RS-232
SSID	Service set identifier, a parameter of Wireless LAN's.	http://en.wikipedia.org/wiki/SSID
SBB	Swiss Federal Railways. German: Schweizerische Bundesbahnen	http://de.wikipedia.org/wiki/SBB
Wardriving	Wardriving is the act of searching for WiFi wireless networks by a person in a moving vehicle, using a portable computer or PDA.	http://en.wikipedia.org/wiki/Wardriving
WiFi	WiFi is the trade name for a popular wireless technology.	http://en.wikipedia.org/wiki/Wifi
WLAN	Wireless local area network	http://en.wikipedia.org/wiki/Wireless_LAN
IC2000	Identification for double decker train cars used by the swiss federal railways.	http://de.wikipedia.org/wiki/IC2000 , in German

1.4. References (Bibliography)

[SUM08TE]	Themeneingabe Master Thesis Thema In-Train-Navigation, 21.8.2008, Marcel Suter. Obtainable from the author.
[SUM08FS]	In-Train Positioning System MAS-06-02.23 Feasibility Study, 10.10.2008, Marcel Suter. Obtainable from the author.
[IEEE98]	IEEE Recommended Practice for Software Requirements Specification, 1998, The Institute of Electrical and Electronics Engineers, Inc. 345 East 47th Street, New York, NY 10017-2394, USA, ISBN 0-7381-0448-5, SS94654 (PDF)
[GPSI08]	Dale DePriest, NMEA data, 2008, http://www.gpsinformation.org/dale/nmea.htm

1.5. Overview

For the current system in use, this SRS contains a description in Appendix A. However, this is not a considered part of the SRS.

For the new system, an overall description and specific requirements are provided.

2. Overall description

2.1. Product perspective

The system will be used, when positioning information in trains is needed. It is intended to replace an older, existing system in a measurement application. The existing system is mostly used in the IC2000 cars of the SBB.



Illustration 1: IC2000 cars of the SBB

In the new system, to input position estimates into the measurement application, a different input interface will be used than currently is. Additionally, also completely different measurement applications will be attachable, via the newly supported interface.

Some technologies for providing the position estimates are required to be used and thus specified as to use.

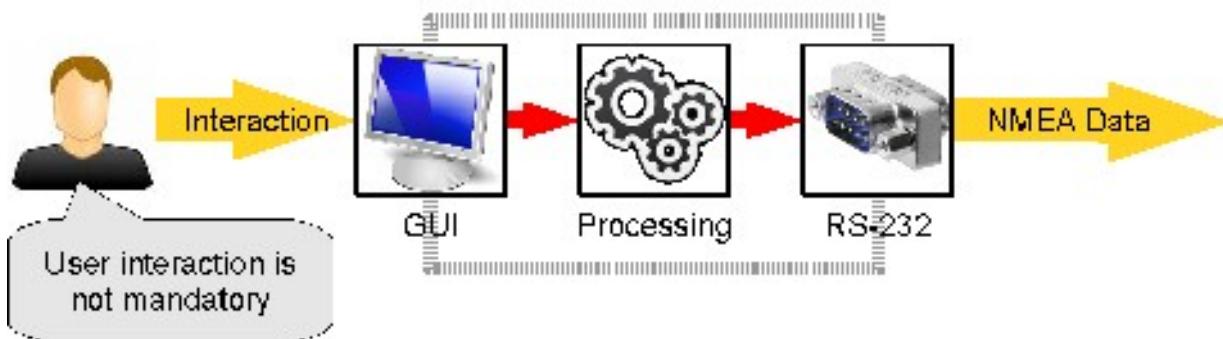


Illustration 2: Overview of external interfaces

2.1.1. System interfaces

The system provides data through the NMEA protocol. Several sensors provide input to the system. All interfaces are specified in Illustration 3.

2.1.2. User interfaces

Via the GUI, the user is able to

- Select the current railway segment
- Fire an event marker and indicate a complete stop of movement
- See status indicators

2.1.3. Hardware interfaces

The system runs on a standard laptop with an RS-232 interface, either built-in or realized via an USB-Adapter.

2.1.4. Software interfaces

The system runs on top of the Windows XP operating system, Service Pack 3 or newer, with the .NET Framework 3.5 or newer.

2.1.5. Operations

The output of the positioning data runs independently of the existence of a GUI. The GUI may be started separately, if at all.

Data provision

Data for the various positioning providers must be available in the formats described below.

2.1.6. Site adaptation requirements

Data Input

The system relies on locally available data. The step of loading the data into the system is only required once for a given set of data.

2.2. Product functions

The system replaces an existing one. An overview over that system is given in Appendix A.

The new system uses various input sensors, whose interfaces are known and specified. The system connects to those interfaces, in order to obtain, process and output positional data. These interfaces define the system boundaries. See Illustration 3.

The data of the sensors (called position providers), gets directly processed in the system itself. The system outputs the resulting position estimates via an RS-232 serial connection in the well known NMEA format. Any attached device may use the position estimates as input.

Since the input (the sensors) and the the output needs to be realized in hardware, there exists a second boundary, the so-called hardware boundary.

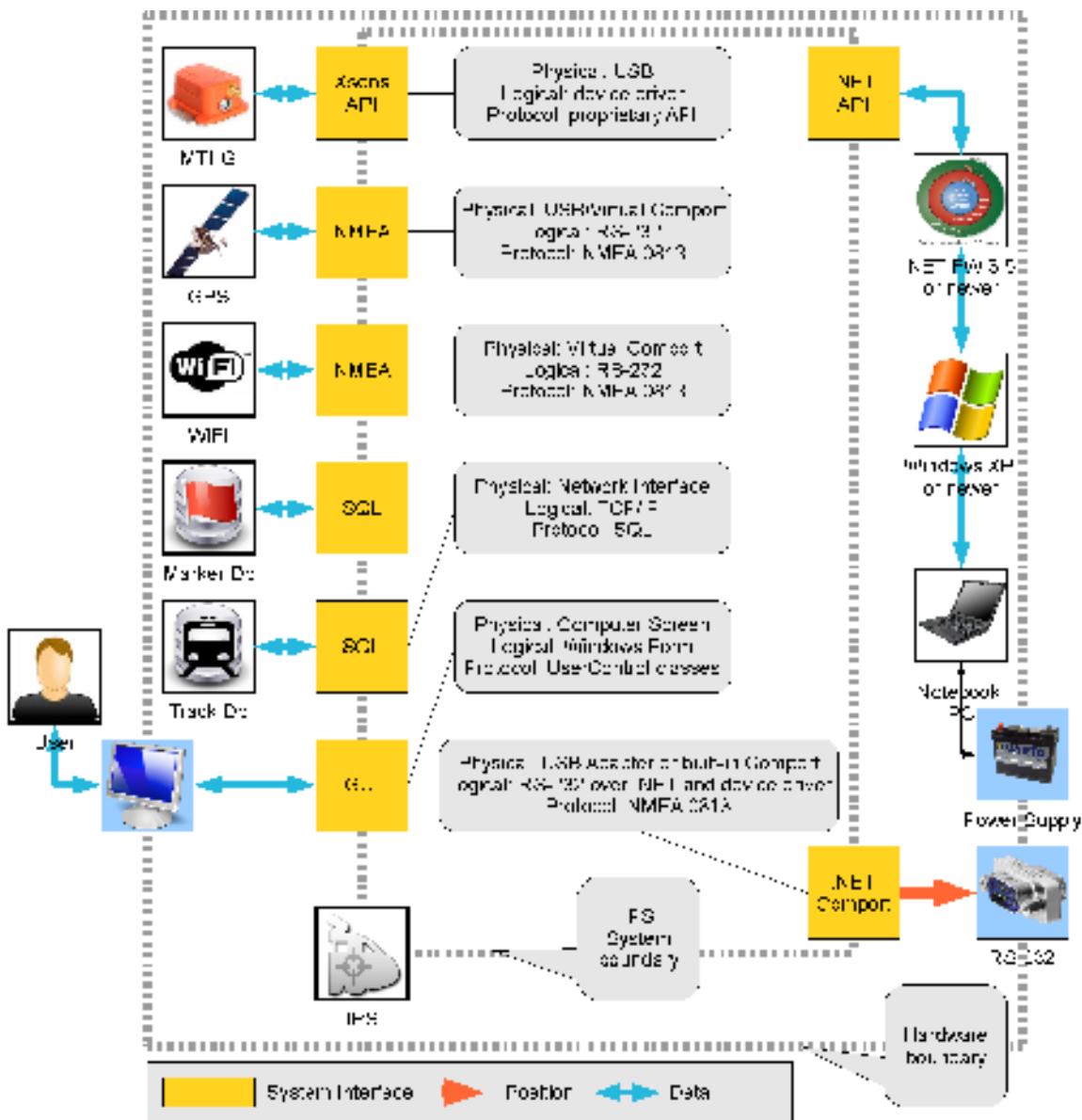


Illustration 3: System boundary diagram

2.3. User characteristics

The average user of the system is a measurement technician that has general computing skills. The user may be occupied with other tasks when using the system.

It can not be assumed, that the user thoroughly understands navigation and positioning methodologies.

No further user characteristic is intended. Especially there is no user identification or login, besides a probably used login on the operating system level. Anyone with access can use the system.

2.4. Constraints

Technologies used for positioning

The system uses multiple technologies from which it derives its current position. These are:

- a GPS receiver with external antenna
- a database of georeferenced WiFi Access Points
- an inertial positioning system
- a track database. The system uses prerecorded data about train tracks to correct the currently determined position from the other position providers.
- An event marker database. The system uses prerecorded event markers and their geographical position to present the user with event markers, he or she can fire.

2.5. Assumptions and dependencies

Power Supply and mechanics

The power supply for the system will not be developed within this thesis. It is assumed, that enough power will be available for operation and there is no power interruption during its uptime. The hardware will only be assembled as prototype.

Look-And-Feel of the GUI

The GUI may be all contained within a single dialog or they may get integrated into several independent visual elements. The exact implementation may be chosen as part of the designed system architecture.

3rd party library usage is allowed

The system depends on the .NET Runtime package, version 3.5, Service Pack 1. It is also assumed, that also any available other libraries may be used in the development. Care should be taken about the quality and available support for the libraries.

As database system, the Microsoft SQL server 2005 or any newer version may get used.

Only basic math is used

Only basic mathematical algorithms are used for the processing of the positioning data. With this assumption, the position estimate is calculated on a best-effort basis.

Availability of data for the input technologies

All data used for the event marker database, the track database and the WiFi Access Point database is already available in an easily parseable file format. Acquiring this data is not part of this thesis.

Where necessary or useful, small amounts of sample data may get acquired during the development of the system.

Installation and Distribution

A simple batch script shall be provided which installs all needed component on a fresh operating system installation. This process may need manual user interaction, for example to get device drivers and the corresponding hardware installed, according to the instructions from the respective manufacturer.

It is assumed, that, besides the components of this system, no other software runs on the same computer.

Position estimate quality

There is no probability check of the outputted position estimates. The position estimates do not emulate the real movement of the train along the track. Merely the position estimates try to provide the most accurate guess of the real location at any time.

For estimating the position, the effect of delay caused by the traveling speed and computation duration is neglected.

Hardware

The used laptop features an internal WiFi card that is accessible using the native Windows API.

As GPS sensor, the EVK-5H Evaluation kit from u-blox is available.

As INS, the MTI-G from Xsens is available.

2.6. Apportioning of requirements

In later versions, the following requirements could be additionally met:

- The processing of position estimates uses advanced mathematical algorithms, as for example the Kalman filter and the Bayes estimator.
- Additional inputs interfaces may be supported.

Overall description

3. Specific requirements

ID	Name	Description	Prio.	
3.1. External interface requirements				
3.1.1. System Interfaces				
1	NMEA Protocol	The well-known NMEA 0813 protocol is used for outputting data from the system.	1	
2	NMEA Update	Each Sentence is issued every second.	1	
3	NMEA serial configuration	The standard NMEA serial configuration is supported, which is 4800 b/s (bit per second rate) with 8 bits of data, no parity, and one stop bit.	1	
4	NMEA speed	Higher baud rates are supported: 9800 b/s and 115200 b/s	2	

Specific requirements

5	RMC Sentence	<p>The RMC Sentence with all parameters is provided. The data is compiled as follows, where the units are according to the NMEA specs:</p> <ul style="list-style-type: none">● Time of fix is the current UTC● Navigation receiver warning is A (valid) when the currently estimated CEP is less than the specified CEP, V (Warning) otherwise.● Latitude and direction is according to the current position estimate● Longitude and direction is according to the current position estimate● Speed over Ground (in Knots!), is current speed estimate● Track angle in degrees is the current track estimate● Date is the current date● Magnetic variation is set to zero● The checksum is calculated according to the data.	1	
---	--------------	--	---	--

Specific requirements

6	GGA Sentence	<p>The GGA Sentence with all parameters is provided. The data is compiled as follows, where the units are according to the NMEA specs:</p> <ul style="list-style-type: none"> ● Time of fix is the current UTC ● Latitude and direction is according to the current position estimate ● Longitude and direction is according to the current position estimate ● Fix Quality indicator. It is set to: 0 (invalid) when no positioning information was yet obtained since system start. 1 (GPS fix) if a fix from the GPS input was available within the last 3 seconds. 6 (estimated) otherwise ● Number of satellites is set to: to the number of satellites from the last GPS fix, if it is not older than 3 seconds, 0 otherwise. ● Horizontal dilution of position is set to an estimation of the CEP of the current position estimate. ● Altitude, Meters, above mean sea level is set to the value of the last GPS input ● Height of geoid (mean sea level) above WGS84 ellipsoid is set to the corresponding value of the last GPS input ● time in seconds since last DGPS update is left empty ● DGPS station ID number is left empty ● The checksum is calculated according to the data. 	3	
---	--------------	---	---	--

3.1.2. User Interfaces

7	GUI not mandatory	The system must also work (with accordingly limited precision) with no user input at all.	1	
56	GUI English	The language of the GUI of the system is US English.	1	

Specific requirements

	GUI translatable	The language of the GUI can be changed without recompilation, e.g. by replacing one or more files.	1		
Selecting the current railway segment					
8	GUI railway segment number	At any time, the user may input the railway segment number he or she is currently traveling on. This narrows down the possible positions to a set of lines between the way points for this segment.	1		
9	GUI segment as text	The input is possible via a text field.	1		
10	Track db input matching	If the input does not match to an entry in the track database, an error message is shown to the user.	1		
11	Track db input plausibility	A plausibility check against the current position is done for this user input.	3		
12	Track db input success	A successful input of a segment number (re)populates the event marker list. As long as the user has not entered a valid segment number, the event marker list remains empty, thus disallowing firing of event markers.	1		
Event marker firing by the user					
13	Event firing	The user must be able to fire an event when traveling past a landmark in the event marker database.	1		
14	Stop event	The user has the possibility to fire an event, when the train is completely stopped.	1		
15	Event list scrollable	The event markers are presented in a scrollable list. Each list entry equals to one event marker.	1		
16	Event list population	After the list gets populated (see above), the first event marker is selected.	2		

Specific requirements

17	Event reselection	When the user fires an event marker, the next one in the list is automatically selected.	2	
53	Event shortcut	Firing the selected event marker is possible using a single shortcut key on the computer keyboard.	2	
54	Event double-click	Firing an event marker is possible by a double-click of the mouse pointer on an event marker in the list.	1	
18	Event re-firing	The system allows a given event marker to be fired more than once. This allows a user to somewhat "correct" a previously executed firing which was too early.	2	
19	Last event	If the last event marker in the list was fired, no entry is selected.	2	
Status output				
20	Status at any time	At any time, the user can see status information of all attached position providers.	2	
21	Position Provider Status information	Each position provider provides the following status information: a free textual description (e.g. "3 Satellites in view"), a standard deviation for the last known position, the timestamp of the last known position, and a status indication from one of the following states. <ul style="list-style-type: none"> ● Working:The provider has issued a new position in the last 3 seconds. ● Ready:The provider is ready to provide positioning information. ● Faulty:The provider is not able to provide positions, because of an error. 	2	
22	Output status information	At any time, the user can see status information of the currently outputted, estimated position. The status information is the same as for a position provider, respectively.	1	
3.1.3. Hardware Interfaces				

Specific requirements

23	Comport Adapter	For the serial RS-232 interface output, a USB-To-RS-232 Adapter shall get used, if the computer does not have a built-in interface.	1	
57	Virtual Comport	The NMEA data is additionally outputted via a virtual com port.	3	

3.1.4. Operations

System start and shutdown

24	Engine start	The user starts the system by starting an engine executable file. No other user action is necessarily needed. This step must be automatable at startup of the underlying operating system by a batch file or similar means.	1	
25	GUI start	The user starts the graphical user interface via another executable file, as he or she sees necessity for.	2	
26	GUI stop	Terminating the user interface does not stop the system's engine executable.	2	
27	Engine stop	The user stops the system by terminating the engine executable's process.	1	

3.2. Functions

3.2.1. Using WiFi as a position provider

28	WiFi scan	The system shall scan for surrounding WiFi access points.	2	
29	WiFi lookup	The system shall look up the found access points in a local database with positions of known Access points. If matches are found, the current position of the system shall be estimated by using simple mathematical algorithms.	2	

Specific requirements

3.2.2. Using GPS as a position provider				
30	GPS usage	The system shall use a GPS device.	1	
3.2.3. Using an IMS as position provider				
31	IMU usage	The system shall use an inertial measurement unit to continue providing position estimate when all other technologies provide no position estimate.	1	
3.2.4. Using a track database for correction				
32	Track db usage	Before outputting, the system shall tie all estimated positions to the nearest point on a given set of tracks.	1	
33	Track as waypoints	The tracks are defined by waypoints.	1	
34	Unique segment number	Each track has a unique segment number for identification.	1	
35	Segment narrowing	The set of tracks is narrowed down according to user input of the current track segment number	1	
3.2.5. Using event markers as position provider				
36	Using event markers	The system shall use the event markers fired by the user as positioning data.	1	
3.2.6. Logging				

Specific requirements

37	Log output	The system produces a log of all outputted data.	1	
38	Log disabling	Disabling this function must be possible by configuration.	2	
39	Log file	The log is written to a text file. The filename contains a timestamp of the current UTC time when the log was started.	2	

3.2.7. Estimating the position

41	Blending inputs	The positioning data gathered from the positioning providers, the IMU, and from the track database is blended together.	1	
42	Position estimation	The blending is done in a way that, at any time, the most probable position is outputted, using basic mathematical algorithms.	1	
43	Estimation CEP	The CEP after blending is 25 meters or below when the train is stopped, at any time.	3	

3.3. Performance Requirements

44	Waypoint count	The system is capable of working with as much as 100000 track waypoints.	1	
45	AP count	The system is capable of working with as much as 100000 stored access points.	2	
46	Maximum speed with GPS	The system works at traveling speeds up to 200 km per hour, when the GPS sensor is providing position estimates.	1	
64	Maximum speed with any sensor	The system works at traveling speeds up to 200 km per hour, when any of the sensors is solely providing position estimates.	3	

3.4. Database requirements

Specific requirements

47	Db update	When the system is not running, all data for the track waypoints, the WiFi Access points, and the event markers is replaceable.	2	
----	-----------	---	---	--

3.5. Design constraints

48	Using laptop	The system runs on a standard, mid-class, laptop computer of a recent date.	1	
49	Using 3 rd party libs	Used 3rd party components and its interfaces must be noted. In case they do not allow commercial usage through their licensing model, they must be later replaceable and the interfaces used must be described.	1	
55	Engine separated	The system has the positioning engine (without any GUI) and the GUI in separate executables.	2	
63	Emulation mode	One or more position provider can get emulated to the system with some form of prepared position estimates. This serves for the test of parts of the system without using the actual hardware or boarding a train. An interface for implementation is provided for this.	1	

3.5.1. Standards compliance

50	NMEA output	The output conforms to the NMEA 0813 protocol	1	
----	-------------	---	---	--

3.6. Software system attributes

3.6.1. Source Code

51	Coding Guidelines	The coding guidelines of the customer are applied.	1	
----	-------------------	--	---	--

Specific requirements

3.6.2. Availability					
52	Availability	The system must be able to run 10 hours continuously.	2		
3.7. Other requirements					
58	Quick Start Guide	A "Quick Start Guide" of 2 pages A4, laminated is provided in English and German. No further user manual is required.	1		
59	Documentation	A documentation according to the needs of the Master Thesis is provided.	1		
3.8. Accuracy determination					
60	GPS Accuracy	The accuracy when using GPS and the track database alone, in open sky conditions, in an IC2000 car, while the train is stopped, is evaluated.	2		
61	WiFi Accuracy	The accuracy when using WiFi and the track database alone, when at least 3 Access Points are visible, in an IC2000 car, while the train is stopped, is determined.	2		
62	Inertial Accuracy	The accuracy when using inertial measurements and the track database alone, when the train is stopped, is determined, one minute after one of the position providers provided an estimate. This may get simulated in the office without actually boarding a train.	2		
65		(intentionally empty)			

Appendix A: Overview of the existing system

Positioning data flow overview

Illustration 7 shows the positioning data flow in the existing system. Various input devices provide absolute and relative information about the current position. This information, together with the measurement data, is stored in one or more measurement files, depending on the current measurement task. The files are transferred manually to a backoffice computer. In the backoffice, an operator re-references the relative positioning information by using the stored event markers. The relative informations gets adjusted according to stored information about the event markers and the tracks. This process requires much user interaction from the operator.

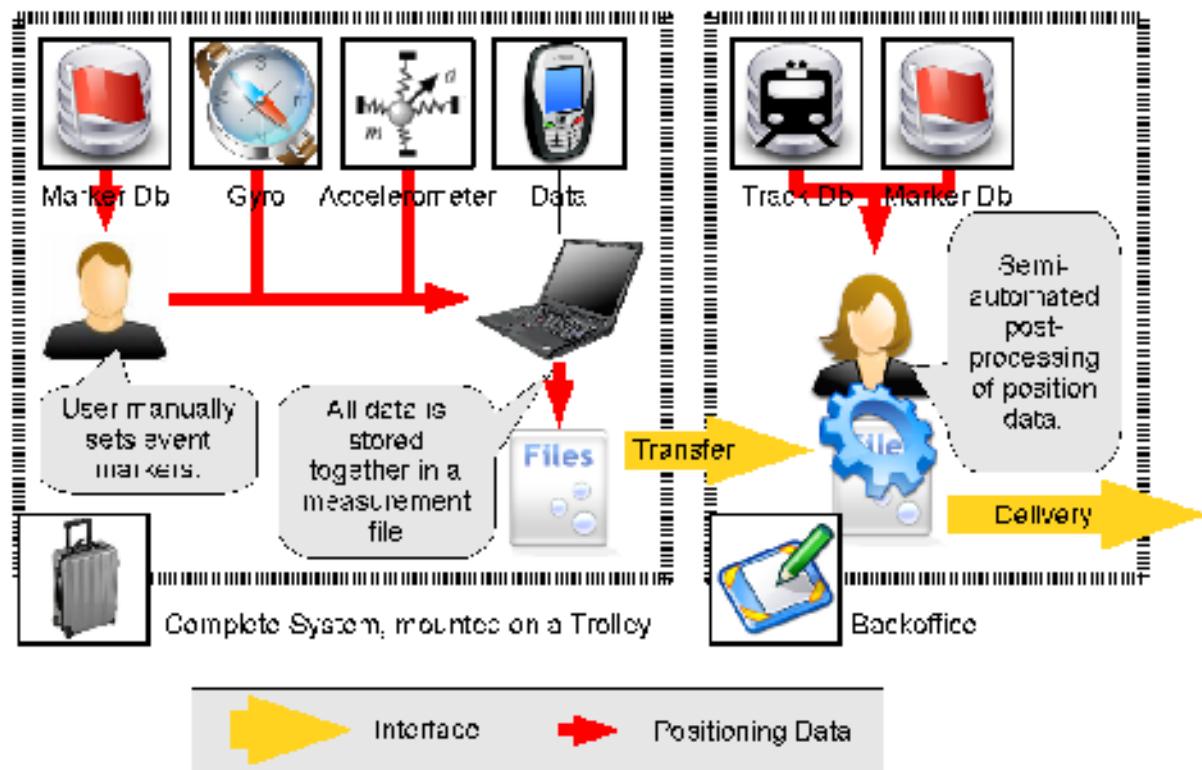


Illustration 4: Positioning information data flow in the existing system

The Hardware

The current solution is built in a trolley. It consists of a single box containing an accelerometer and a gyro.

Appendix A: Overview of the existing system



Illustration 5: The current measurement solution is built on a trolley

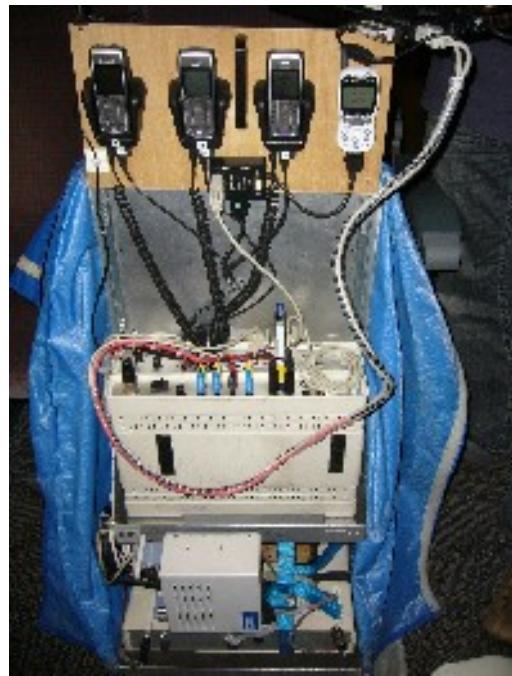


Illustration 6: A look inside the existing measurement trolley, laptop removed. The inertial system is on the bottom.

A laptop is used to measure the radio parameters and to read out the data of the gyro and the accelerometer via an RS-232 serial connection. All Positioning and measurement data is then saved into the measurement files. However, no real positioning data is saved; just the accelerometer and gyro data are saved in a customized format for later post-processing.



Illustration 7: The measurement technician at work

Appendix A: Overview of the existing system

The currently used laptop is an IBM Thinkpad R52. It has 8GB and a 18GB Hard disk in it. The processor runs at 1.5GHz and there is 512MB RAM.

The software

The GUI of the current positioning solution is built into the GUI of the measurement software.

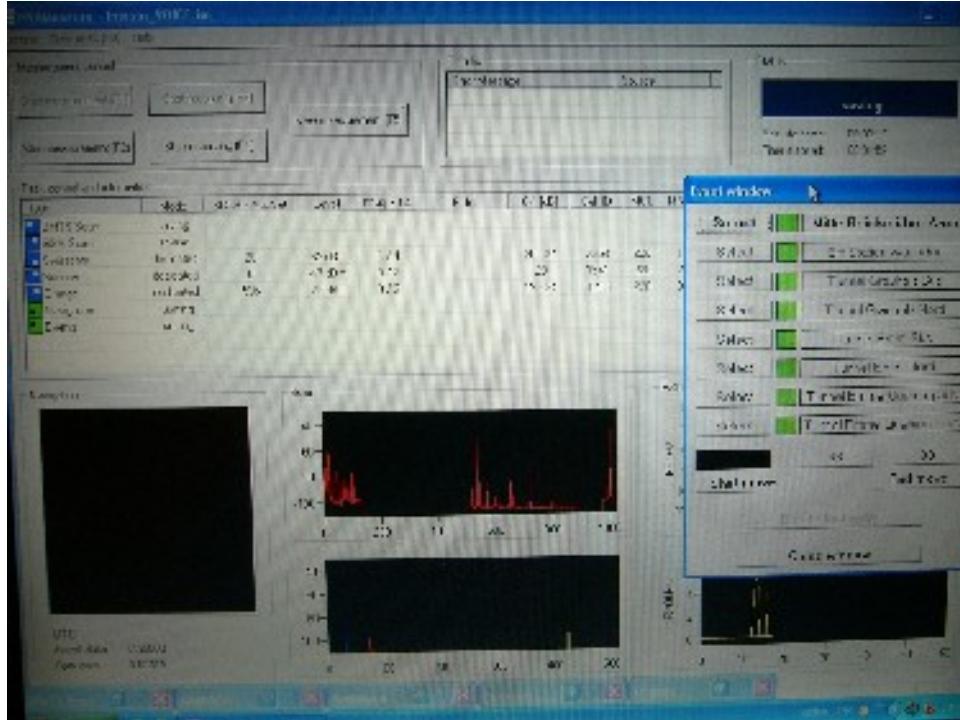


Illustration 8: GUI of the current GSM measurement solution, measurement is running

When starting the measurement, the user inputs the railway segment number of the current segment he or she is traveling on into the system. The system looks up the number in local database and presents a set of predefined event markers denoting landmarks along the track.

During the course, the user clicks on each of this marker when it passes. Additionally two buttons can be pressed to indicate either that the train stopped moving or that it just has started moving. This and the event markers are later used in the post processing step, to correct the stored inertial data.

The user has the possibility to revoke a marker up to 5 seconds after it has been fired. This is useful in case of errors when a landmark was misinterpreted.

Additionally the user can make comments during and after the course to indicate specific events or circumstances experienced during the measurements, be it of positioning matter or else. (Applies only to the GSM version of the measurement software, in the UMTS version, comments are only allowed at the start of the measurement).

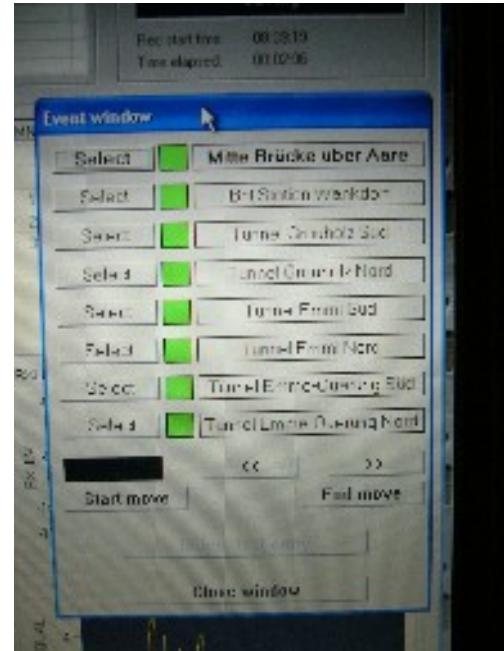


Illustration 9: (Detail) Available event markers during the course

Measurement modes

There are 4 possible measurement modes:

- GSM Voice
- GSM Data
- UMTS Voice
- UMTS Data

The laptop is set up as dual boot system with two Windows XP operating systems. Each system is set up for either measuring with the UMTS or the GSM Hardware. Also the software used, is different.

The pictures shown here are all with the GSM Hard- and Software. In terms of positioning, the requirements will be the same for GSM and UMTS measurements. Both systems take the data from a serial comport using the RS-232 protocol.

Quantities, Uptime and power

During a normal day, up to 8 files, each representing a measurement along one railway segment are created. During a year, measurement is done on 50 days.

The system runs on a 12 Volt Accumulator of 60Ah capacity. It may be continuously up for 10 hours, including the laptop. The laptop is usually not shut down during the day, not even when changing the train.



In-Train Positioning System

Master Thesis



Main Documentation

Abstract Design and implement a multi-input positioning system for use in trains.

ID MAS-06-02.23

Keywords GPS, WiFi, INS, SQL, WCF, train, C#, .NET

Author Marcel Suter
Kanalweg 3, CH-3125 Toffen
+41-(0)76-387 70 06
marcel.suter@gmx.net

Supervisor Stefan Bigler, ENKOM Inventis AG
+41-(0)31-950 42 42
stefan.bigler@enkom.com

Expert Rolf Wenger, Infobrain AG
Rolf.Wenger@infobrain.com

Class MAS-IT 2006-02

Date 19. Feb. 2009

*For everyone who asks receives;
he who seeks finds;
and to him who knocks, the door will be opened.*

The Bible in Luke 11,10 (NIV)

Revision History

Tag	Date	Author
First draft	9.10.2008	Marcel Suter
Added Use Cases	3.11.2008	Marcel Suter
Added Implementation Chapter, Added Component Diagram, Changed and Improved Use Cases, Rearranged the Table diagram of the RailRouteDb, Improved Sequence diagram for Db Access, Added Analysis of the MTi-G INS.	12.1.2009	Marcel Suter
Added implementation details about aligning	6.2.2009	Marcel Suter
Finishing Documentation for Version 0.9	17.2.2009	Marcel Suter

Management Summary

The Master Thesis "In-Train Positioning" showed the possibilities of a multi-input positioning system. Most project goals have been met, and the following inputs are used

- High sensitive GPS
- Accelerometer with compass
- User input with event markers
- Alignment with a track database

Not included in the solution is the intended use of georeferenced WiFi Access Points.



Illustration 1: Outline of the In-Train Positioning System

This document is based on the feasibility study for this Master Thesis, in [SUM08FS] and the SRS, in [SUM08SR]. It describes the work undertaken to fulfill the SRS.

It consists of all the usual steps of a software project:

- Analysis
- Synthesis
- Design
- Implementation
- Test

At the end, currently known issues are mentioned. Also, an outlook to further improvements and extensions is given.

Total time used for the Master Thesis is approximately 410 hours.

Table of Contents

Management Summary.....	4
1 Analysis and Synthesis.....	8
1.1 Mind map.....	8
1.2 Use Cases.....	8
1.2.1 Introduction.....	8
1.2.2 Actors.....	8
1.2.3 Use Case diagram.....	8
1.2.4 Use case descriptions.....	9
1.3 Engine process overview.....	14
1.4 Deployment diagram.....	15
1.5 Component diagram.....	15
1.6 GUI.....	16
1.7 Position providers.....	17
1.8 The position Information.....	17
1.8.1 Units.....	18
1.9 Activity Diagrams / Scenarios.....	19
1.9.1 User starts the engine to get NMEA data (UC Start System).....	19
1.9.2 Acquire and process position estimates.....	20
1.9.3 Emitting the current position.....	21
1.9.4 The GPS or Place Lab are providing a position estimate.....	22
1.9.5 The INS is providing a position estimate.....	23
1.9.6 The User is providing a new position estimate.....	24
1.9.7 Merging Process.....	24
1.10 Configuration of the engine.....	26
1.11 Translation between HDOP and CEP.....	27
1.12 Error correction of the INS.....	27
1.13 Find nearest point on track.....	30
1.14 Extrapolation of positions.....	31
1.14.1 Algorithm.....	31
1.15 Existing Database.....	33
1.16 Data relations.....	35
1.16.1 MARKER_DEFINITION.....	35
1.16.2 RAIL_STATION.....	35
1.16.3 RAIL_STATION_STAT.....	35
1.16.4 ROUTE_DATA.....	35
1.16.5 ROUTE_DEFINITION.....	35
1.16.6 ROUTE_DEFINITION_ROUTE_DATA.....	36
1.16.7 ROUTE_ORDER.....	36
1.16.8 ROUTE_SET.....	36
1.16.9 ROUTE_SPEZ.....	36
1.16.10 ROUTE_SPEZ_ROUTE_DATA.....	36
1.16.11 MAPINFO.MAPINFO_MAPCATALOG.....	36
1.17 Tables to use.....	36
1.18 Database Access.....	36
1.19 Initial position.....	38
1.20 Accuracy and Drift of the MTi-G INS.....	38
1.20.1 Accuracy.....	38
1.20.2 Drift.....	40
1.21 Integrating the Acceleration on a single axis.....	42
2 Design.....	43
2.1 Package Diagram.....	43
2.1.1 Diagram.....	43

Management Summary

2.1.2 Package description.....	43
2.2 Engine core.....	44
2.2.1 Business class diagram.....	44
2.2.2 Example Object diagram.....	47
2.2.3 Sequence diagram for building the chain.....	48
2.3 Data Exchange.....	49
2.3.1 Using WCF.....	49
3 Implementation.....	52
3.1 Overview.....	52
3.2 Changed packaging.....	52
3.2.1 GuiWinAppExec.....	53
3.2.2 Test.....	53
3.2.3 Common.....	54
3.2.4 Core.....	54
3.2.5 RailrouteDbAccess.....	54
3.2.6 GuiLib.....	54
3.2.7 CmtComWrapper.....	54
3.3 Calculating the heading deviation for GPS data.....	54
3.4 Tools.....	54
3.4.1 Interfaces.....	55
3.4.2 Abstract base classes.....	56
3.5 Using virtual serial ports.....	59
3.6 Checking the CEP calculation for GPS.....	59
3.7 Implementing the INS provider.....	60
3.7.1 Using the SDK from XSens.....	60
3.7.2 Configuring the MTI-G.....	60
3.8 Representing global coordinates.....	61
3.9 Implementation of the database access.....	61
3.9.1 Interface.....	61
3.9.2 Designing the database context.....	61
3.9.3 Getting the track segment numbers.....	62
3.9.4 Getting the nearest point on the tracks.....	66
3.10 Class Estimate.....	72
3.11 Class DeviantDouble.....	73
3.12 Class Merger.....	74
3.13 Icons.....	74
3.14 The GUI.....	75
3.14.1 Select Track Segment.....	75
3.14.2 Fire Event Markers.....	75
3.14.3 Status.....	75
3.14.4 Ergonomics.....	76
3.15 Tool chain.....	76
3.16 Configuration.....	78
3.17 External Code and Libraries.....	78
4 3rd Party component usage.....	79
4.1 Crystal Icons.....	79
4.1.1 License.....	79
4.2 u-blox Evaluation Kit.....	79
4.2.1 License.....	79
4.3 NMEA Sentence Interpreter.....	79
4.3.1 License.....	79
4.4 C# Geodesy Library for GPS.....	79
4.4.1 License.....	79
4.5 XSens SDK.....	79
4.5.1 License.....	80

Management Summary

5 Test.....	81
5.1 Unit Tests.....	81
5.1.1 Application.....	81
5.1.2 Unit tests in the solution.....	81
5.1.3 How to test events.....	82
5.2 Test the fulfillment of the Requirements.....	82
5.2.1 Summary.....	86
5.3 Test drives.....	86
5.3.1 Drives.....	86
5.3.2 Detected problems and behavior.....	95
5.3.3 Conclusion.....	96
6 Project Management.....	97
6.1 Review Meetings.....	97
6.1.1 Kick-Off, 9.10.2008.....	97
6.1.2 Meeting on 11.11.2008.....	97
6.1.3 Meeting on 16.12.2008.....	97
6.1.4 Meeting on 13.1.2009.....	97
6.1.5 Meeting on 11.2.2009.....	97
7 Known Issues.....	99
7.1 Currently known Issues.....	99
7.2 Further improvements in the code.....	99
8 Lessons learned.....	101
9 Future.....	102
9.1 Optimizations	102
9.1.1 GUI.....	102
9.1.2 AHRS and Kalman.....	102
9.1.3 GPS.....	102
9.1.4 Improve Acceleration measurements.....	102
9.2 Extensions to the sensors.....	102
9.2.1 Sensors (Position providers).....	102
10 Bibliography.....	104
11 Thanks.....	105
Appendix A: Mindmap of the analysis.....	106

1 Analysis and Synthesis

1.1 Mind map

To gain an overview, I have drawn a mind map of the most important requirements, see Appendix A.

1.2 Use Cases

1.2.1 Introduction

A Use Case Model describes the proposed functionality of a new system. A Use Case represents a discrete unit of interaction between a user (human or machine) and the system. This interaction is a single unit of meaningful work, such as Create Account or View Account Details.

Each Use Case describes the functionality to be built in the proposed system, which can include another Use Case's functionality or extend another Use Case with its own behavior.

1.2.2 Actors

In this thesis, various types of actors do exist. One is the typical user that interacts with the system, another is an external system (typically a measurement application) that uses the output of this system, shown here as position consumer.

Additionally there are actors that represent each sensor.

1.2.3 Use Case diagram

The use case diagram in Illustration 5 shows the use cases from the user and the external system's point of view.

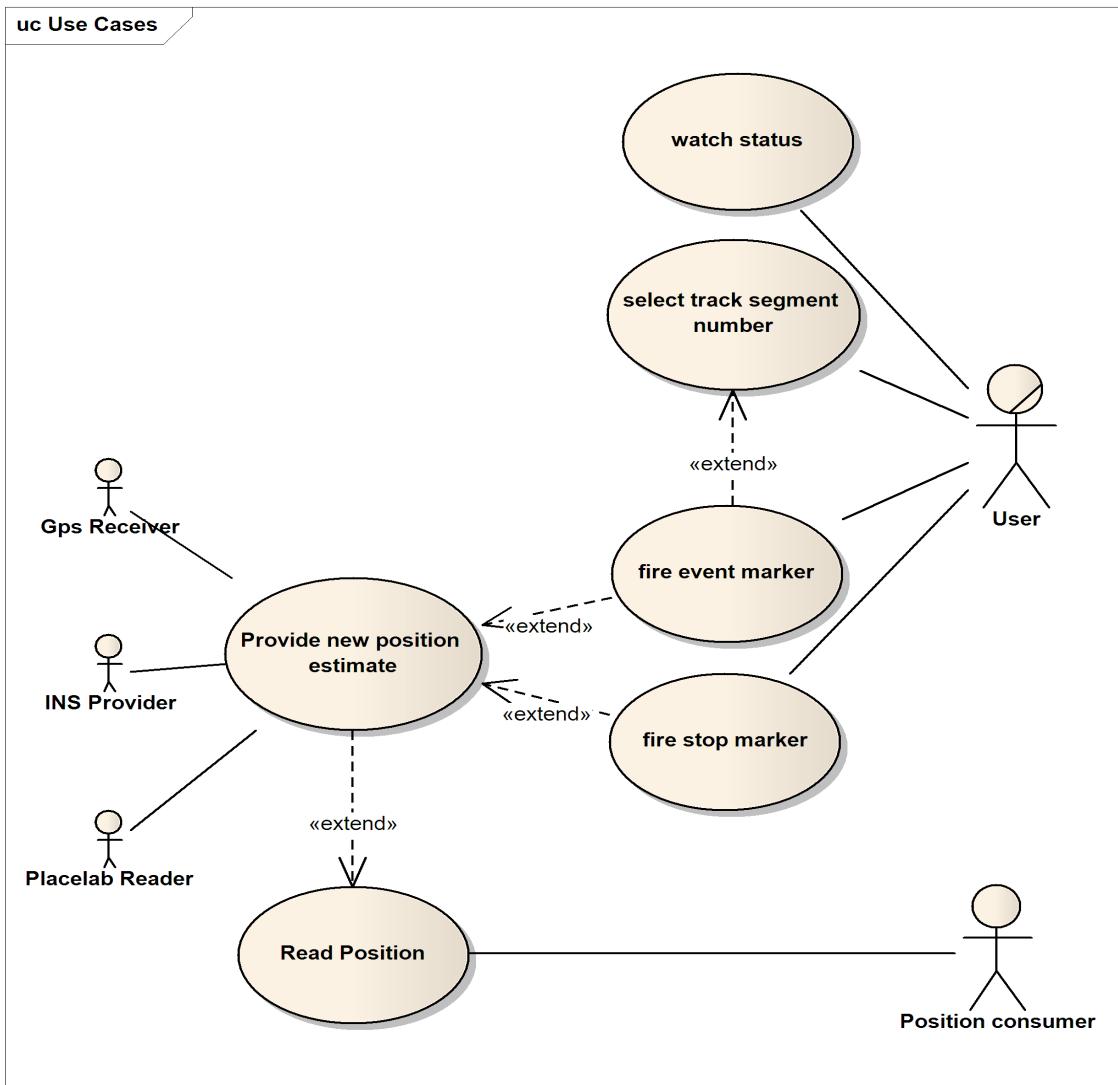


Illustration 2: Primary use cases

The most important use case is “Read Position”. This invokes a cascade of other use cases. When the position consumer wants to read the position, there must be a position available beforehand. This is modeled in the use case “Provide new position estimate”. Here, the various position provider, be it a sensor, or the user via the GUI, can provide their position information. The user does this through two dedicated use cases, “fire event marker” and “fire stop marker”. Also, before an event marker is able to get fired, a track segment number must have been set. While all this happens, the user may watch the status of the system via the “watch status” use case.

1.2.4 Use case descriptions

<i>Read position</i>	
Description	The main usage for this system. The position is outputted once per second by the system.
Actor	Position consumer
Precondition	Engine is started.
Activities	The current best position estimate is made available for reading via the RS-232 interface.
Postcondition	One position estimate per second is received
Invariant	No system failure, no power interruption.
Incoming data	No external data needed.
Outgoing data	Current position estimate, in the NMEA protocol.
Comments	If no position is yet acquired by the sensors, the last estimate from previous runs may get outputted.
Requirement numbers	1,2,3,4,5,6
<i>Select track segment number</i>	
Description	The user inputs a track segment number
Actor	User
Precondition	GUI and engine started.
Activities	<p>The user inputs the track segment number he or she is currently traveling on in a input field. The system afterwards looks up all event markers for that segment in the database. It presents the event markers in a list in the GUI, so the user may fire them.</p> <p>The possible waypoints for the position enhancement is narrowed down to the selected track.</p>
Postcondition	<p>The event marker for this segment are presented to the user.</p> <p>The position enhancement for the nearest point on track works with the selected segment.</p>

Analysis and Synthesis

Invariant	
Incoming data	User-provided track segment number.
Outgoing data	None outside the system
Comments	
Requirement numbers	8,9,10,11,12

Fire event marker

Description	The user fires one of the presented event markers.
Actor	User
Precondition	Use case "select track segment number" is carried out. GUI is started, engine is started.
Activities	The position of the selected event maker is transferred to the engine, and gets processed. The corresponding position provider in the engine issues a position estimate for inclusion in the current best position estimate.
Postcondition	The current best position estimate is enhanced.
Invariant	
Incoming data	User's decision on whether to fire the event.
Outgoing data	No external data.
Comments	
Requirement numbers	13,15,16,17,53,54,18,19

Fire stop marker

Description	The user fires a stop marker
Actor	User
Precondition	None.
Activities	The information that the train is stopped is transferred to the engine, and gets processed. The corresponding position provider in the engine issues a position estimate for inclusion in the current best position estimate.
Postcondition	The current best position estimate is

Analysis and Synthesis

	enhanced.
Invariant	
Incoming data	User's decision on whether to fire the event.
Outgoing data	No external data.
Comments	
Requirement numbers	14

Watch status

Description	The user sees a graphical representation of the current status of the system.
Actor	User
Precondition	Engine and GUI is started.
Activities	The GUI registers itself on the engine and provides a form of callback. The engine updates the GUI when a status does change.
Postcondition	User has seen any status change.
Invariant	
Incoming data	System status from the engine's internal components.
Outgoing data	Visual representation of the status.
Comments	
Requirement numbers	20,21,22

Provide new position estimate

Description	Any of the attached sensors provides a new position estimate.
Actor	User, GPS receiver, INS Provider, Place Lab reader
Precondition	System is started.
Activities	This contains the most part of the business intelligence. Any incoming position estimate is merged with others, extrapolated in time if necessary, and stored in an internal storage as current best position estimate.

Analysis and Synthesis

Postcondition	The system has digested the new position estimate. Future emitted estimates reflect the new estimate. Status information is updated as applicable.
Invariant	The system uninterruptedly continues to output the current best position estimate.
Incoming data	Position estimate from one of the sensors.
Outgoing data	None directly.
Comments	The actors act individually, there is no assumable timely correlation between the provided estimates. Each estimate from each actor may affect the system somewhat differently.
Requirement numbers	13, 14, 29, 30, 31

1.3 Engine process overview

The main task of the engine is to process the position estimates from the various providers. Illustration 3 shows a possible design.

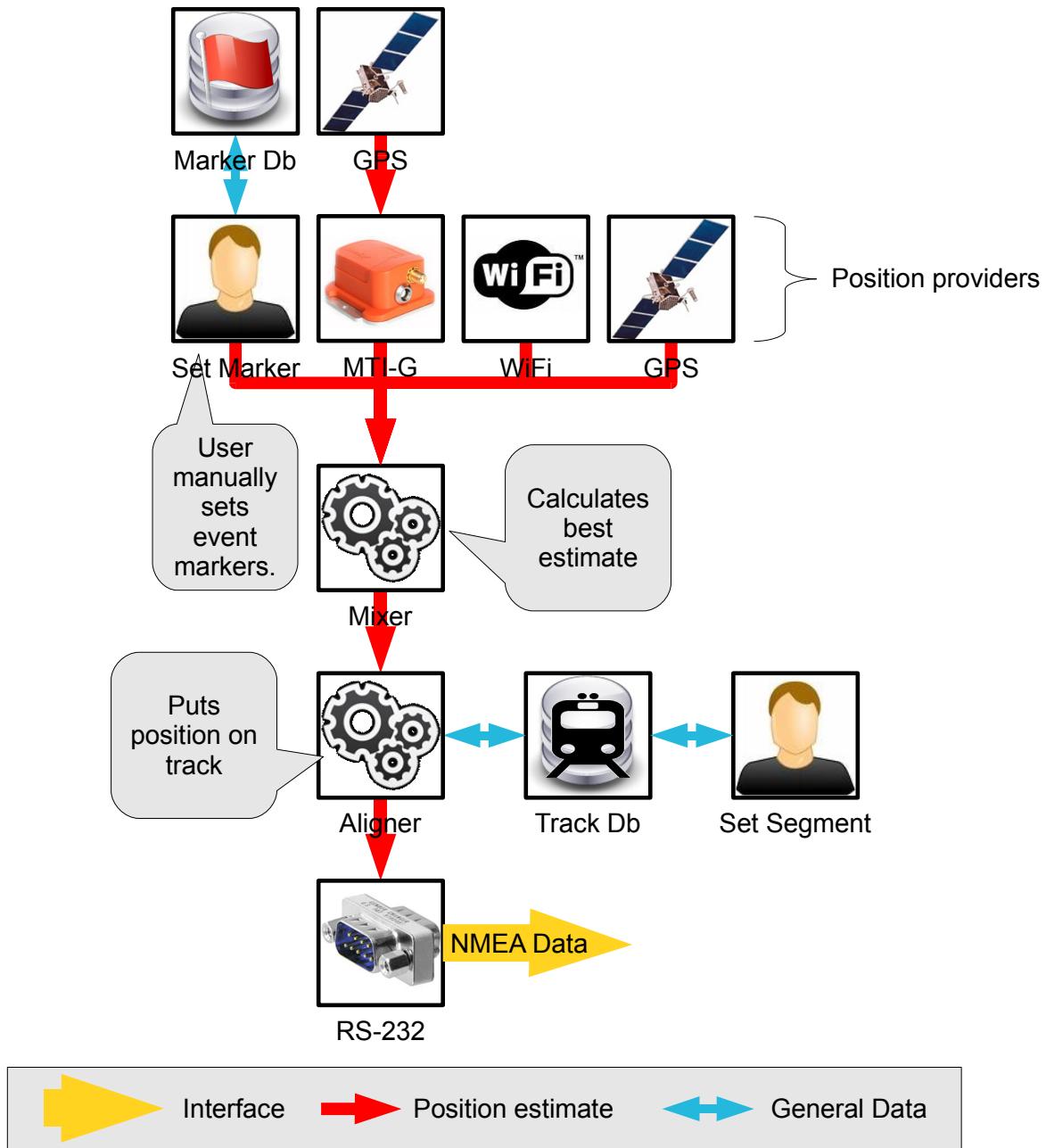


Illustration 3: Engine process overview

1.4 Deployment diagram

The deployment diagram shows the separation of the processes, and how they could get hosted on various connected computers. Hosting on different machines is however not required, and most probably also not used if implemented, but shown here to make the separation of the components clear.

The deployment diagram shows also, that the GUI component does not have direct access to the database. This helps creating the dependencies small and clearly structured.

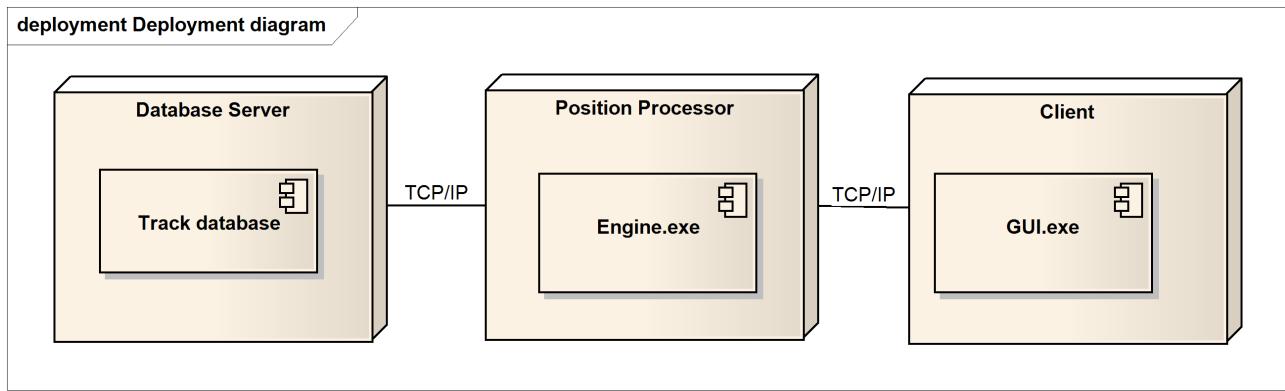


Illustration 4: Deployment diagram

1.5 Component diagram

The component diagram shows the main components of the system with their interfaces. The interfaces shown will get implemented explicitly. Some of the interfaces may be within the same package from the Illustration 4.

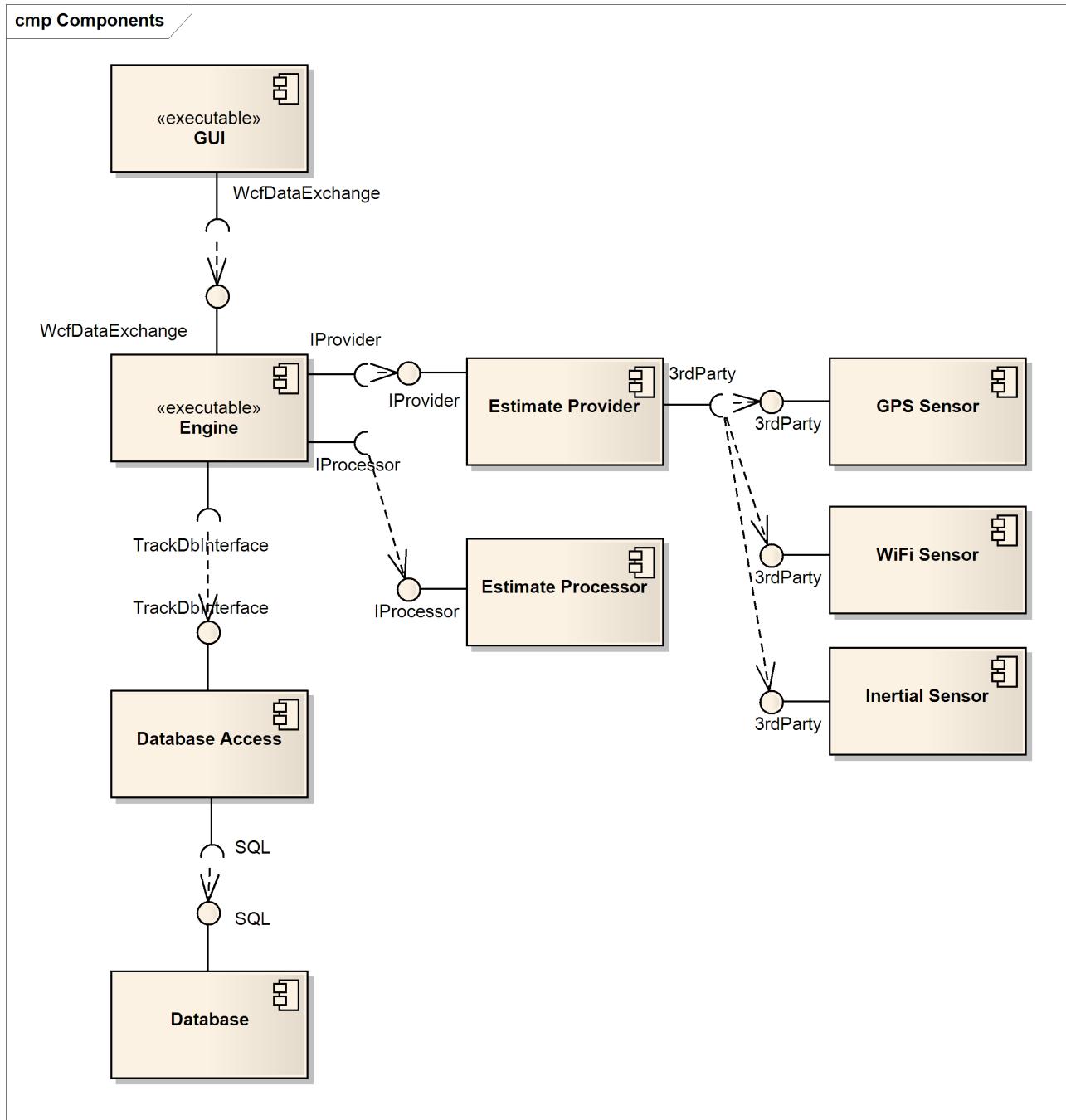


Illustration 5: Component diagram

1.6 GUI

A possible GUI is shown in Illustration 6. All requirements from the SRS are included, but the actual design may still vary. Especially the Event Marker firing and the status indication may get visually improved.

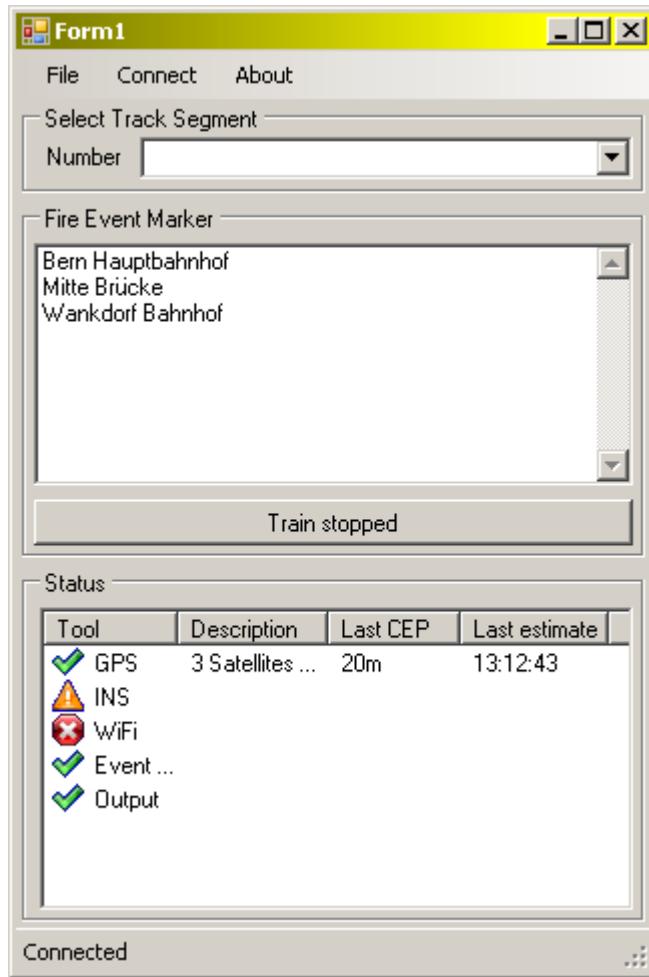


Illustration 6: GUI prototype

In this prototype, the status is shown as colored icons besides the tool name.

1.7 ***Position providers***

Position providers are units that provide position estimates. Each instance of a position provider, or at least each type of, must have a unique identifier, to allow a processed position estimate to declare it's origins.

1.8 ***The position Information***

The whole system does mainly process positioning estimates. Such an estimate should contains the following information, based on the requirement #5 and #6:

A note about time: The GPS time is not perfectly in sync with UTC, because of reasons discussed in an article by Joe Mehaffey¹. An overview of time differences can be found on leapsecond.com². Most GPS receivers seem to take the gap into account, and thus I take no specific measures against time differences. The time of a position estimate should always be the UTC time as returned by the .NET Framework.

Stored data

1 See: <http://gpsinformation.net/main/gpstime.htm>

2 See <http://leapsecond.com/java/gpsclock.htm>

- Time and Date in UTC, at that the position is actually estimated. This may also be in the future or in the past, depending on the acquisition time and on what has happened or will probably happen to the estimate.
- estimated Latitude
- estimated Longitude
- estimated Speed (horizontally)
- estimated Heading (horizontally)
- estimated Acceleration (in the direction of the heading only)
- Origin (This is a dictionary of the unique identifiers of the position providers, from which the Latitude and Longitude values are taken from, with percentage of their contribution to the position).
- Magnetic variation. (This may get taken from a GPS input, and is not mandatory by requirement. It could be however useful anyway)
- Fix Quality indicator, an enumerated value
- Number of satellites
- estimated Altitude
- Height of Geoid³ (can get taken from a GPS reading, if available)

All values that are estimated must also contain a standard deviation. For Latitude/Longitude, the CEP is given, because a standard deviation is not useful here.

It is also possible for a position estimate, to contain only some of the above mentioned values, while the others are omitted. Especially the acceleration will probably only get used partially, since it is not outputted via the NMEA protocol.

1.8.1 Units

I recommend using metrical units where not otherwise required. This would result in the following unit usage:

- Horizontal position (Latitude and Longitude): Degrees with decimal places
- Vertical position (Altitude): Meters
- Distances on the surface: Meters
- Speed: Meters per Second
- Acceleration: Meters per Second per Second
- Horizontal moving direction (Heading): Degrees with respect to true north⁴
- Fix Quality Indicator: the values are according to the specification
 - 0 = invalid
 - 1 = GPS fix (SPS)
 - 2 = DGPS fix
 - 3 = PPS fix
 - 4 = Real Time Kinematic
 - 5 = Float RTK
 - 6 = estimated (dead reckoning) (2.3 feature)

3 See <http://en.wikipedia.org/wiki/Geoid> for an explanation

4 See http://en.wikipedia.org/wiki/True_north and http://en.wikipedia.org/wiki/North_Pole for an explanation.

7 = Manual input mode
8 = Simulation mode

NMEA Protocol

The NMEA protocol uses different units in their protocol. They must get translated from and to the above when using it.

Latitude/Longitude: Two digits for the integer value of degrees, then a decimal representation of the seconds, with usually 3 digits after the decimal point.

Speed is measured in knots, where 1 Knot = 0.5148 Meters per Second⁵

1.9 Activity Diagrams / Scenarios

1.9.1 User starts the engine to get NMEA data (UC Start System)

This is probably the most basic activity and will surely be broken into many classes in the design phase. It shows the basic process of acquiring and emitting positional data.

5 Source: <http://www.chrismanual.com/Intro/convfact.htm>

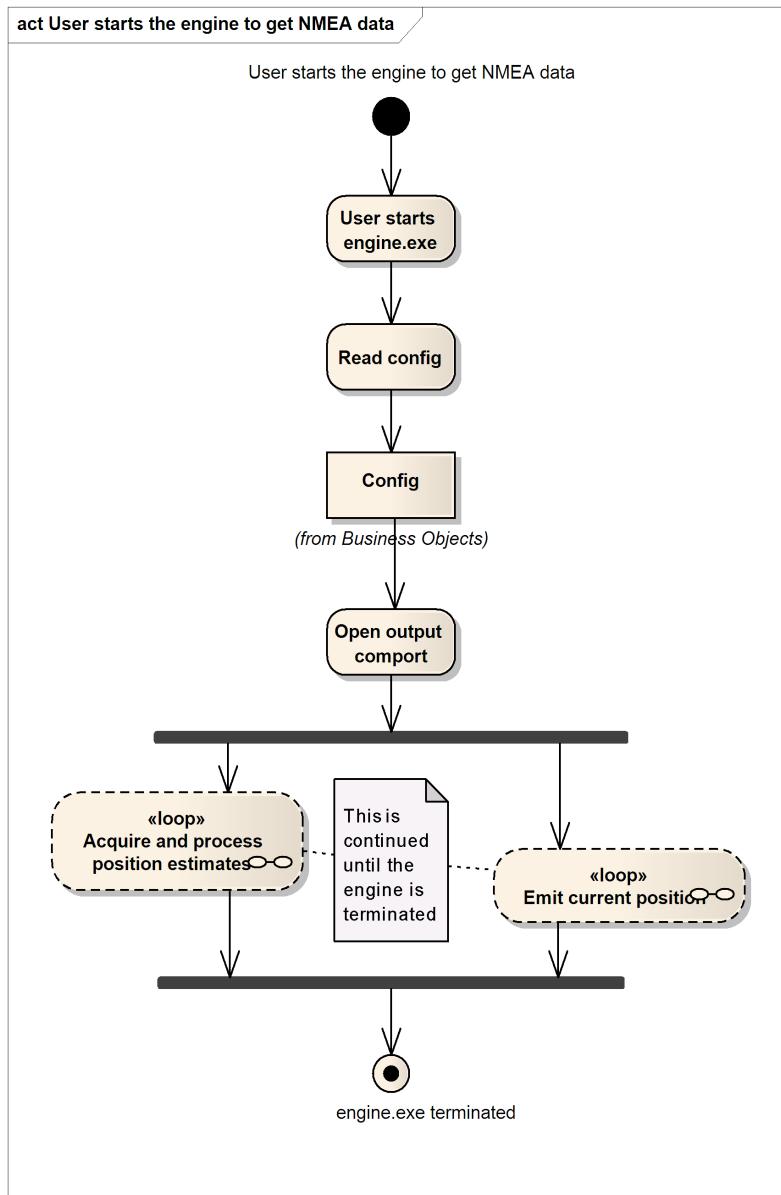


Illustration 7: Activity diagram for "User starts the engine to get NMEA data"

The activity "Acquire and process position estimates" is shown here as a loop. In practice, this could well be fully event-driven. This activity is analyzed in more detail in the following chapters.

1.9.2 Acquire and process position estimates

This is a structured sub-activity from Illustration 7.

The newly available estimates are acquired form the various sensors and then processed.

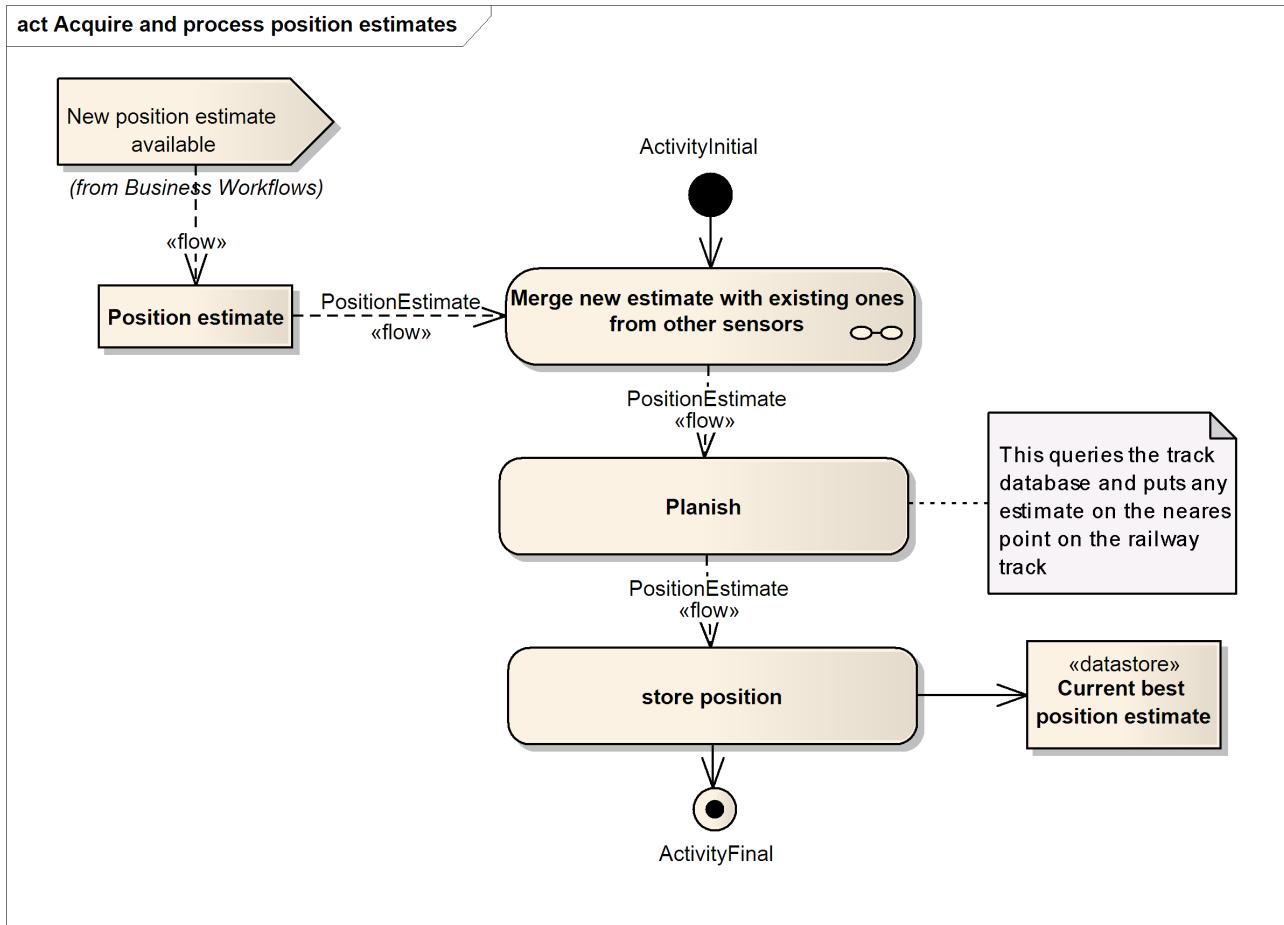


Illustration 8: Activity diagram "Acquire and process position estimates"

The acquisition is shown for each sensor in later sub-chapters.

1.9.3 Emitting the current position

This is a structured sub-activity from Illustration 7. Its task is to take the current best position estimate and send it to a position consumer.

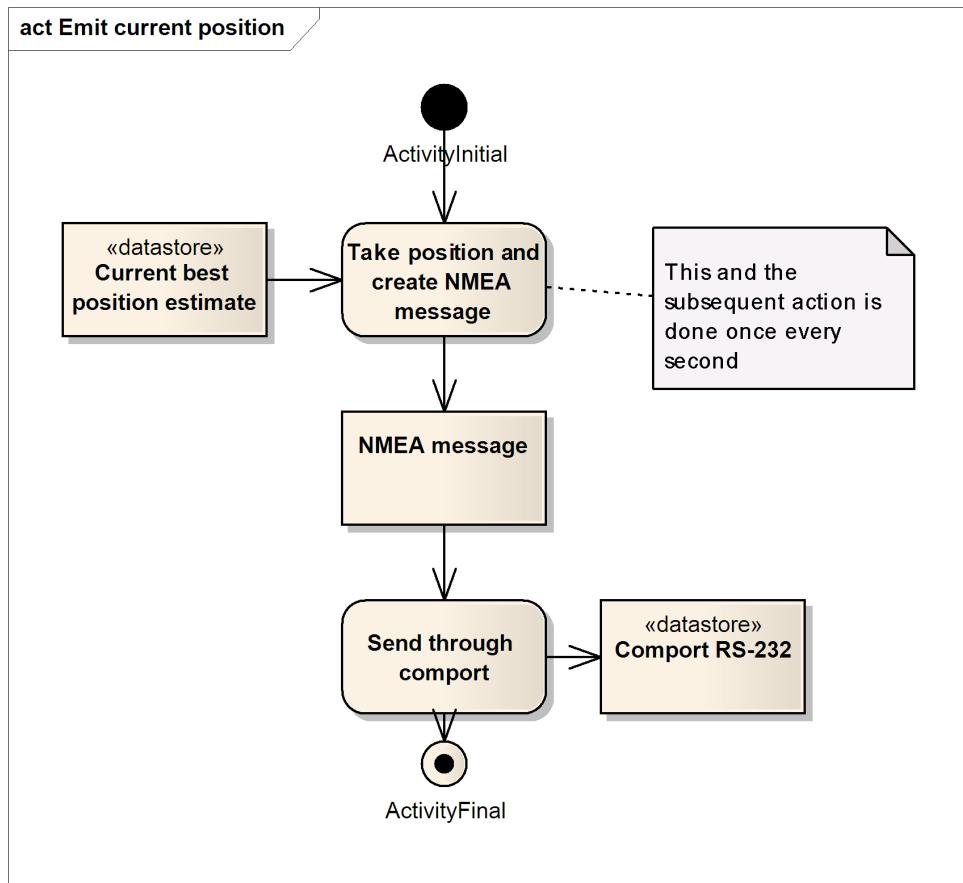


Illustration 9: Activity diagram "Emit current position"

1.9.4 The GPS or Place Lab are providing a position estimate

I assume that there is a component readily available, that reads the comport and parses the NMEA sentences in its own thread, and that the received data is made available through a .NET event. If such a component is not available, it has to be designed and implemented separately.

Since Place Lab can provide its position estimates via NMEA, we use this interface for simplicity, even though it is not ideal from a communication and efficiency point of view.

At the end, the position estimate is made available through an event.

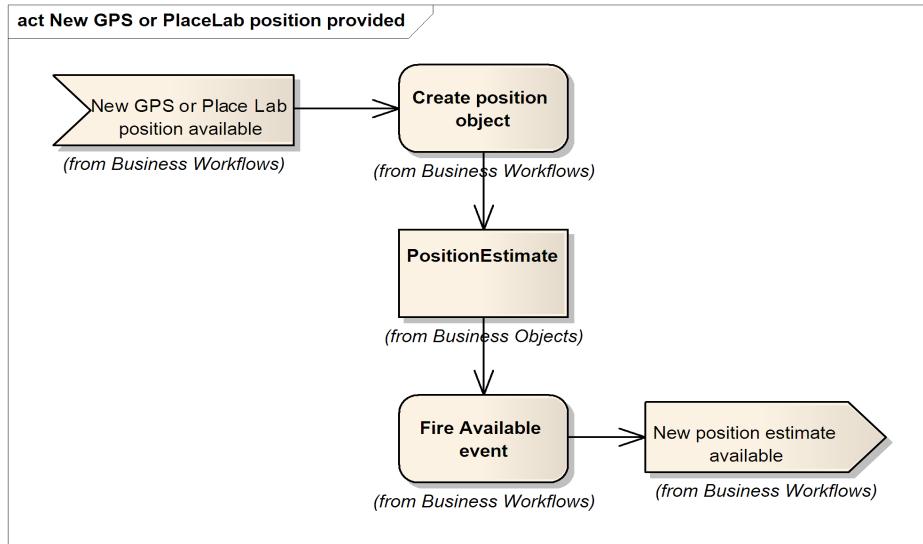


Illustration 10: Activity diagram for "The GPS or PlaceLab are providing a position estimate"

1.9.5 The INS is providing a position estimate

The XSens API provides a DLL for accessing the sensor data. Currently, the exact specification for the DLL is not known, so I assume, that there will be a polling mechanism instead of events. See also the user manual [XSNS08UM].

At the end, the position estimate is made available through an event.

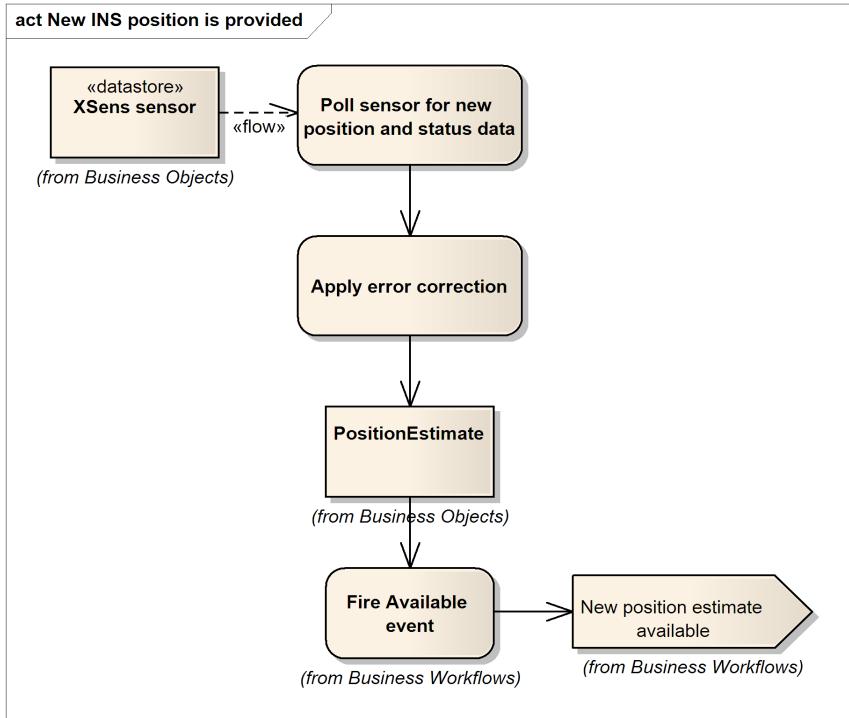


Illustration 11: Activity Diagram for "The INS is providing a position estimate"

1.9.6 The User is providing a new position estimate

This also applies for when the user is firing the train stopped event. In this case, the position estimates contains no coordinates, but speed and acceleration values are set to zero.

At the end, the position estimate is made available through an event.

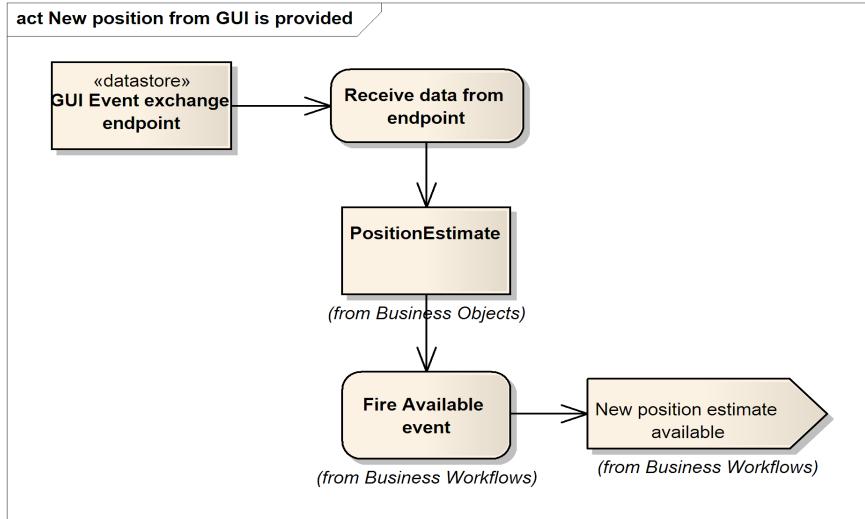


Illustration 12: Activity diagram for "New position from GUI is provided"

1.9.7 Merging Process

Literature

There is plenty of literature available on the theme of calculating the most probable position of vehicles. One of the most related articles to my problem is [PLST08AE]. However, since only basic mathematical algorithms are required to use, this is beyond the scope.

Own solution

The main problem when merging is, that not all position providers have an estimate available at the same time. However, to provide an accurate merged position estimate, time-based distortions (jitters) must get eliminated.

The basic concept to solve time jitters is to use extrapolators, that extrapolate the last position estimate of each source to the time now.

From the time-update position estimates, I calculate a weighted average of their position information. I use the reciprocal value of the CEP as weighting factor. Averaging is done by weighting all coordinates (latitude and longitude), sum them then divide through the sum of the weights. The result is the new average.

The new CEP is the averaged CEP from the used extrapolated estimates.

Illustration 13 shows the process in a graphical diagram.

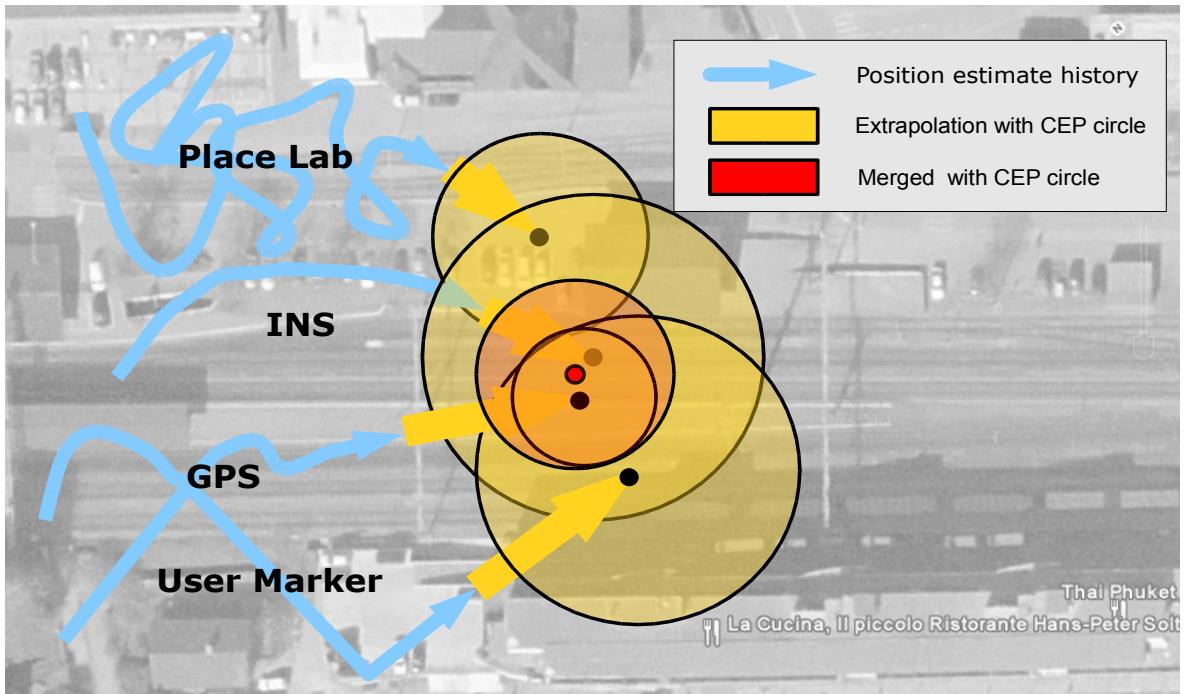


Illustration 13: Graphical overview of merging

Illustration 14 shows the process as activity diagram.

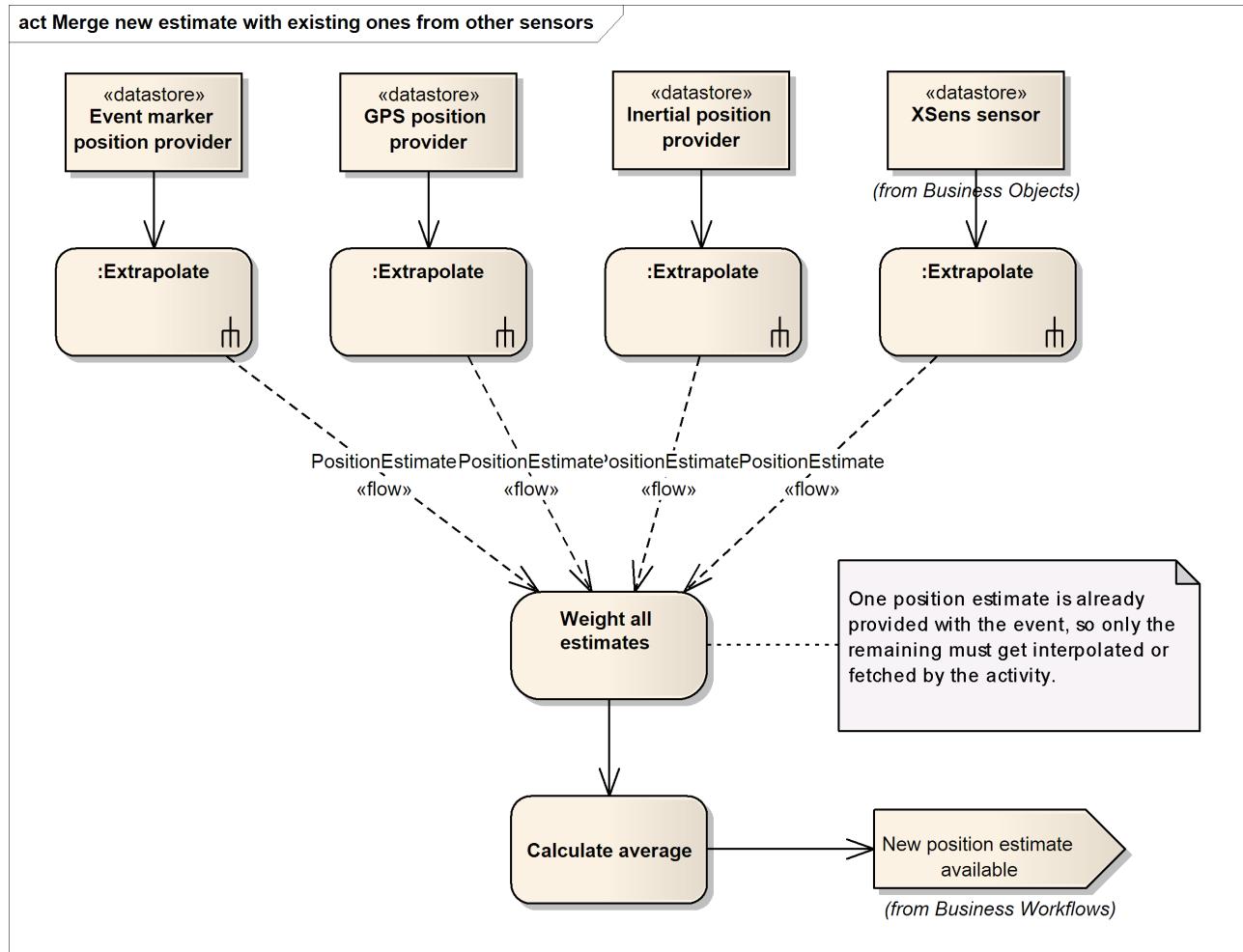


Illustration 14: Activity diagram for the merging process

This process is done for each new position estimate provided by any provider. Thus, the merger class will need to have explicit access to the providers (or to the extrapolators if they are classes for themselves) to get extrapolated estimates

1.10 Configuration of the engine

The configuration defines vital parameters of the system. It will not be user changeable and be static for one system, once set at installation time. The configuration will provide the following data:

- Comport for the GPS receiver NMEA data
- Comport to use for the Place Lab NMEA data
- Comport to use for position estimate emitting
- Default CEP for a fired event marker
Since the accuracy of the user input is limited, all position estimates from the GUI are tagged with this CEP value
- Default CEP for Place Lab data
The position estimates from Place Lab are not rated with a CEP, so we must assume a value

- Maximum acceleration of the system
This may serve for a probability checking feature, so it can prevent the data output from having large positional glitches.

1.11 Translation between HDOP and CEP

The NMEA protocol defines the usage of HDOP as indication of horizontal precision. Internally, however, the system uses CEP. This requires a conversion between the two units. The HDOP, or DOP in general, is only defined by vague ratings, whereas the CEP is a strict mathematical representation of probabilities. In an article on The Code Project, Jon Person proposes a formula for conversion, under the presumption of a specific accuracy of the GPS device⁶. We could use the formula as:

$$e_{cep} = 6 * e_{hdop}$$

Formula 1:
Conversion
between HDOP
and CEP, where
the accuracy of
the GPS device
is assumed to
be 6 meters

A test with the GPS-Receiver should prove the result.

1.12 Error correction of the INS

Error parameter calculation

The INS is of limited accuracy over time and provides no existing means of updating or initializing the position from outside. The only real-world-reference is the built-in GPS. Since we will have other sensors that may provide accurate fixes even while the internal GPS is not having a sufficient GPS signal reception, we want to feed these fixes into the INS position estimates.

The intention is to use a static and dynamic (over time) error correction algorithm. Illustration 15 shows the principle:

6 Source: <http://www.codeproject.com/KB/mobile/WritingGPSApplications2.aspx>

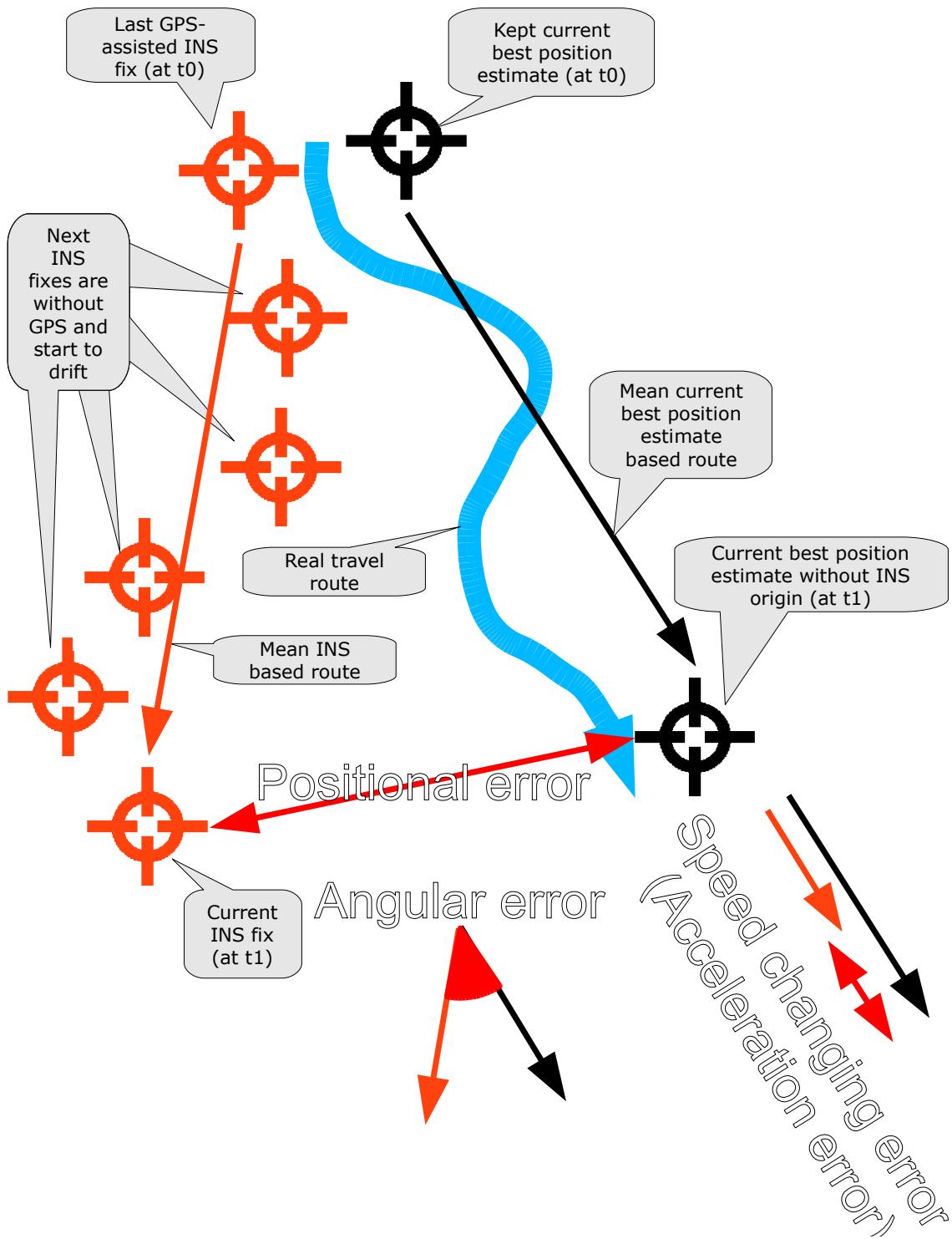


Illustration 15: Calculation of the current error parameters

- When, and as long as, the INS has a good fix with its built-in GPS (at time t_0), we keep this fix plus the current best position estimate of the system (at the same time t_0).
- When, and as long as, the built-in GPS has no fix, continue to output the current INS fix with the error correction applied (values are initially zero)

Analysis and Synthesis

- When, later (at t1), a current best position estimate is available in the system, that is not based on INS and has a better CEP (lower value) than the current INS fix, keep this plus current INS fix of this time.
- Calculate the difference between the INS fix at time t1 and the Current best position estimate at time t1.
This is the current static positional error of the INS unit.
- Calculate the mean heading of the route between the INS fix from t0 and t1.
Calculate the mean heading of the route between the Current best position estimate at t0 and t1.
The difference is the current static angular error of the INS unit.
- Calculate the changing of the speed of the INS fix at t0 and t1. Divide the result by t1 minus t0. This is the current mean INS acceleration.
Calculate the changing of the speed of the current best position estimate at t0 and t1. Divide the result by t1 minus t0. This is the current mean best position estimate acceleration.
The difference is the current mean acceleration error of the INS unit.
- As long as there is no GPS fix from the built-in receiver, apply a correction for all errors to the INS-provided position. The kept current best position estimate from t1 is used as base point for all subsequent corrections.

Illustration 16 shows this algorithm as state event diagram.

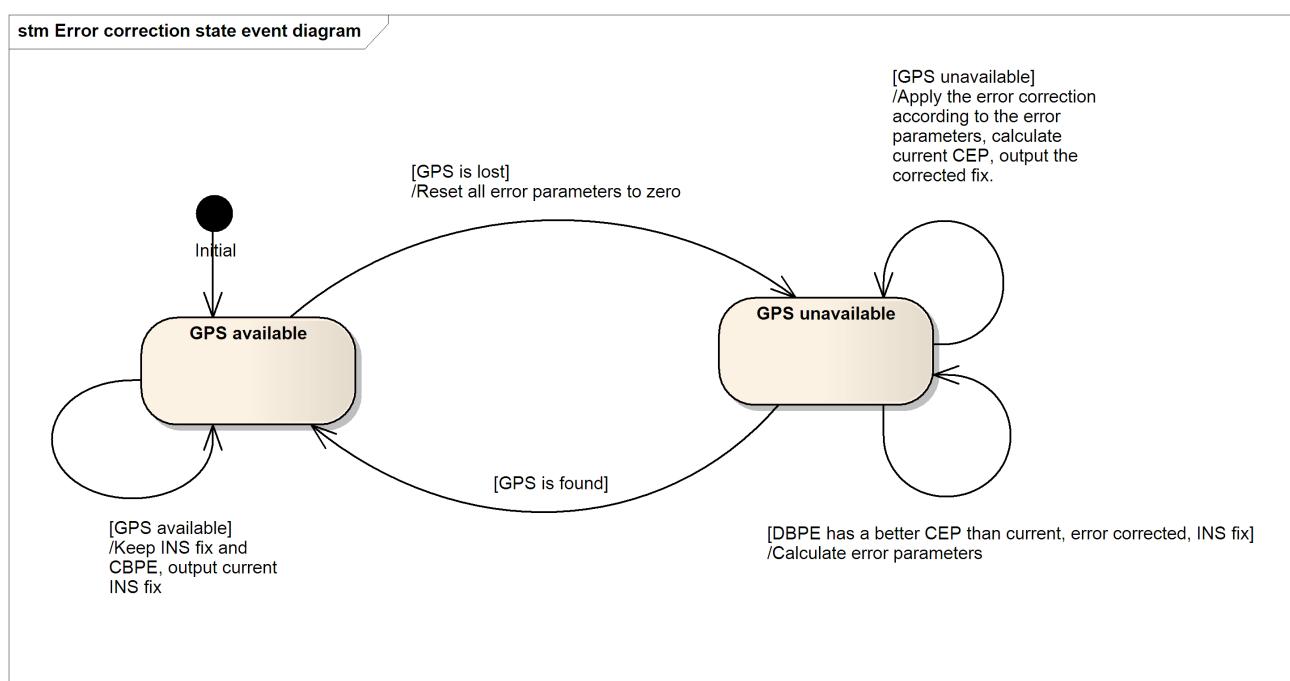


Illustration 16: State event diagram showing the INS error correction algorithm

Error correction

Illustration 17 shows the application to calculate the error correction parameters for the time t_1+x :

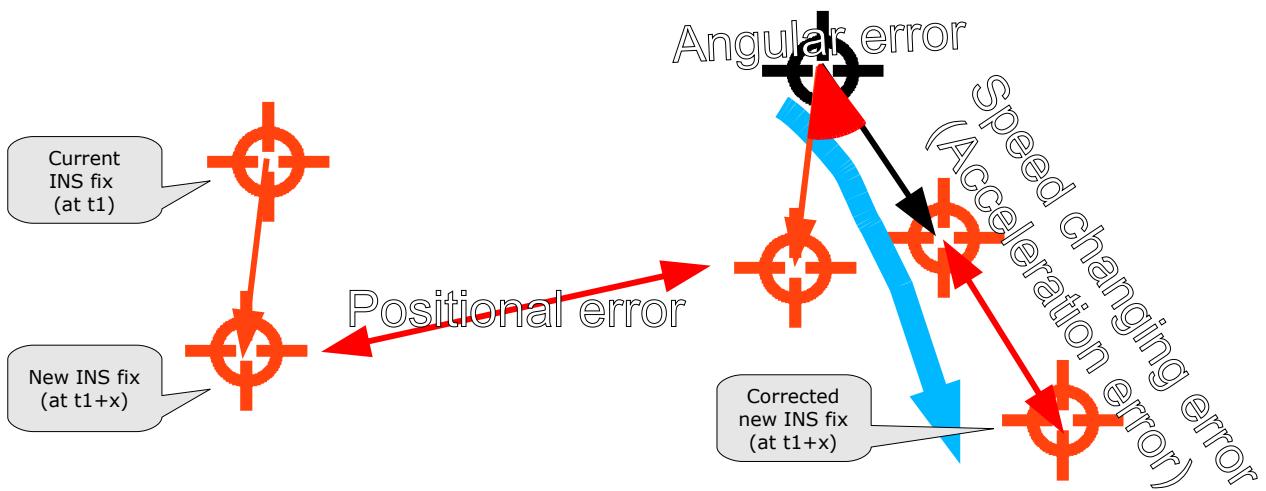


Illustration 17: INS error correction

- The positional error is corrected by applying the negated value of the positional error.
- The angular error is corrected by “walking” back the path traveled since time t_1 . Then the same distance is “walked again in the direction with the angular error removed.
- The acceleration error is corrected by calculating the equivalent erroneous distance since t_0 and applying it to the current position from above. The erroneous distance is the squared delta time between t_1 and t_1+x , multiplied with the acceleration error. See the formula below:

$$d_e = ((t_1 + x) - (t_1))^2 * e_a$$

Formula 2: Calculation of the erroneous distance for the acceleration error

1.13 Find nearest point on track

Analysis of the existing database has shown that for each stored waypoint, there is also a heading value in radians available. This simplifies the finding of the nearest point, since now, a single point is enough to align a given position.

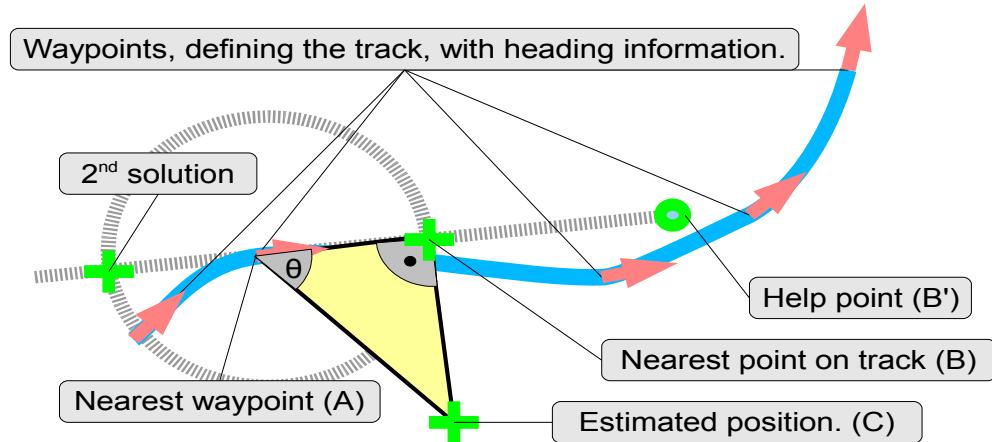


Illustration 18: Finding the nearest point on a track, when the waypoints have heading information.

The algorithm may work as follows

- Find nearest waypoint to the estimated position in the set of all waypoints.
 - Calculate the angle phi between a line in direction of the heading of the waypoint (A-B') and the segment from the waypoint to the estimated position (A-C).
- $$\text{phi} = \arccos\left(\frac{\mathbf{AB}' \cdot \mathbf{AC}}{|\mathbf{AB}'||\mathbf{AC}|}\right)$$
- where AB is the vector from A to B and AC is the vector from A to C.
- Using the angle and it's cosine, calculate the distance from A to the nearest point on track B.
$$AB = \cos(\text{phi}) * |\mathbf{AC}|$$
 - Calculate the two possible solutions using the distance A-B, and find the resulting point that is closest to the estimated position C.

Calculating a point-segment distance

The previously discussed algorithm uses the dot product. This and other basic geometry algorithms are explained by [TOPC08IB]. There is also a code example.

1.14 Extrapolation of positions

Usually, position estimates are issued by the providers on a regular basis, but this is not guaranteed. However, the output of the system is required to provide a position estimate exactly every second. This requires, that the positions in the system must get extrapolated in the meantime.

This is where the estimator feature is needed. Basically, this is a mathematical algorithm, that, based on a history, predicts the future. As long as the time between the last real update and the time to predict to is small, also the mean error is small.

1.14.1 Algorithm

A possible algorithm could be as below. The algorithm does not take changing accelerations and changing of the heading into account. This could later be improved, if necessary.

Illustration 19 shows the main steps in the extrapolation algorithm.

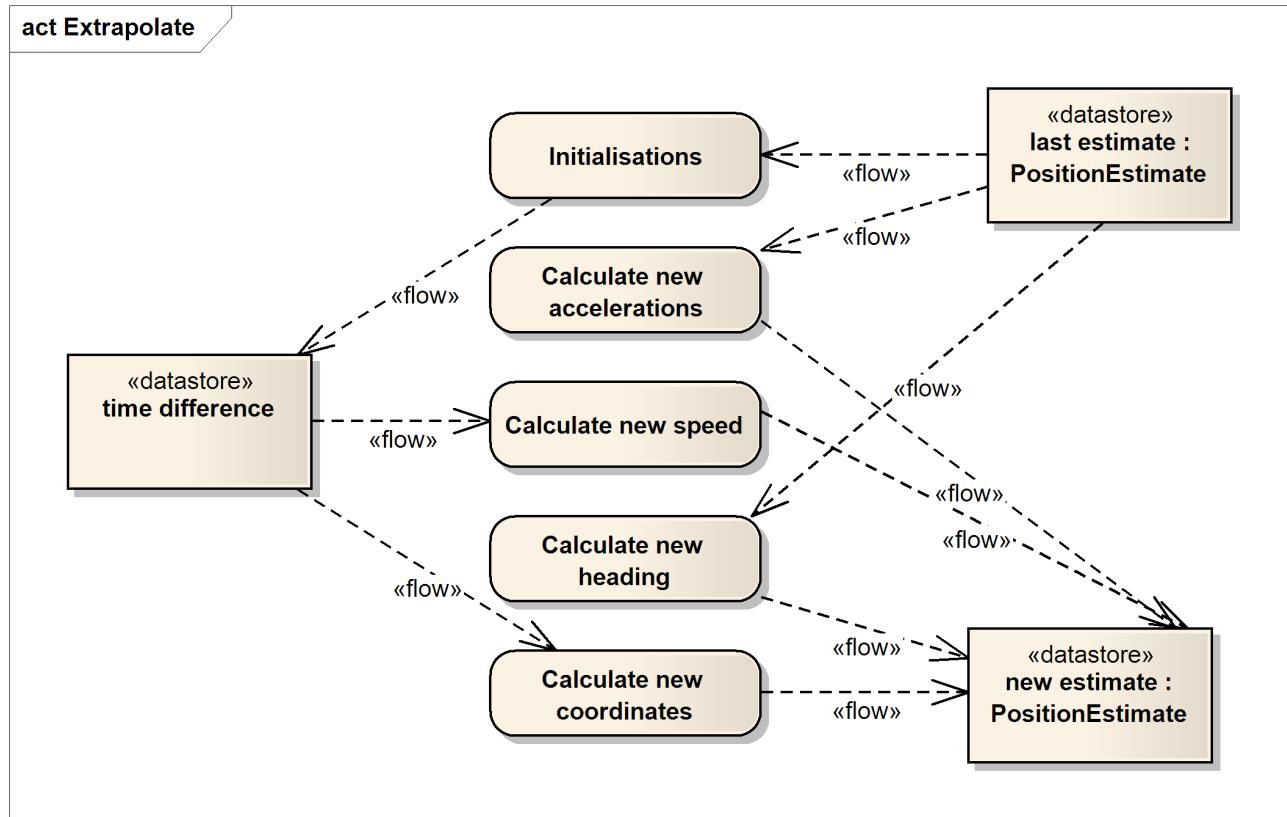


Illustration 19: Activity diagram for extrapolation of estimates

A description of the single actions in more detail is found below:

Initializations

- Consider a last estimate at a given time and a new estimate for the current time, which is to be extrapolated.
- Calculate the time difference between the last estimate and now in seconds.

Calculate new acceleration

- Assign the acceleration to the new estimate. It is assumed, that the acceleration itself does not change, so it is set to the value of the last estimate, or zero if not provided. The standard deviation for the acceleration should get set to the value of the last estimate too, but, if not provided I recommend setting it to 0.1 meters/second/second. This is 10% of a considered maximum acceleration for trains.⁷

Calculate new speed

- Take the assigned acceleration of the new estimate and multiply it by the time difference. This is the change of the speed. Add this change of speed to the speed of the last estimate and use as speed of the new estimate.
- Do the same as above for the acceleration standard deviation, to calculate the new speed standard deviation.

Calculate new heading

- It is assumed, that the heading does not change. Assign the heading of the last estimate to the new estimate.

⁷ Source: <http://www.physicsforums.com/archive/index.php/t-73565.html>

Calculate new coordinates

- Take the speed and heading of the new estimate. Build a vector and multiply it with the time difference. This results in a position translation. Add the translation to the position of the last estimate.
- Do the same as above for the speed standard deviation, to calculate the new CEP.

1.15 Existing Database

I have found, that all needed data already exists in a MSSQL database. Thus, instead of creating my own database and importing the data from text file, I will use this database.

The definition is as follows:

Analysis and Synthesis

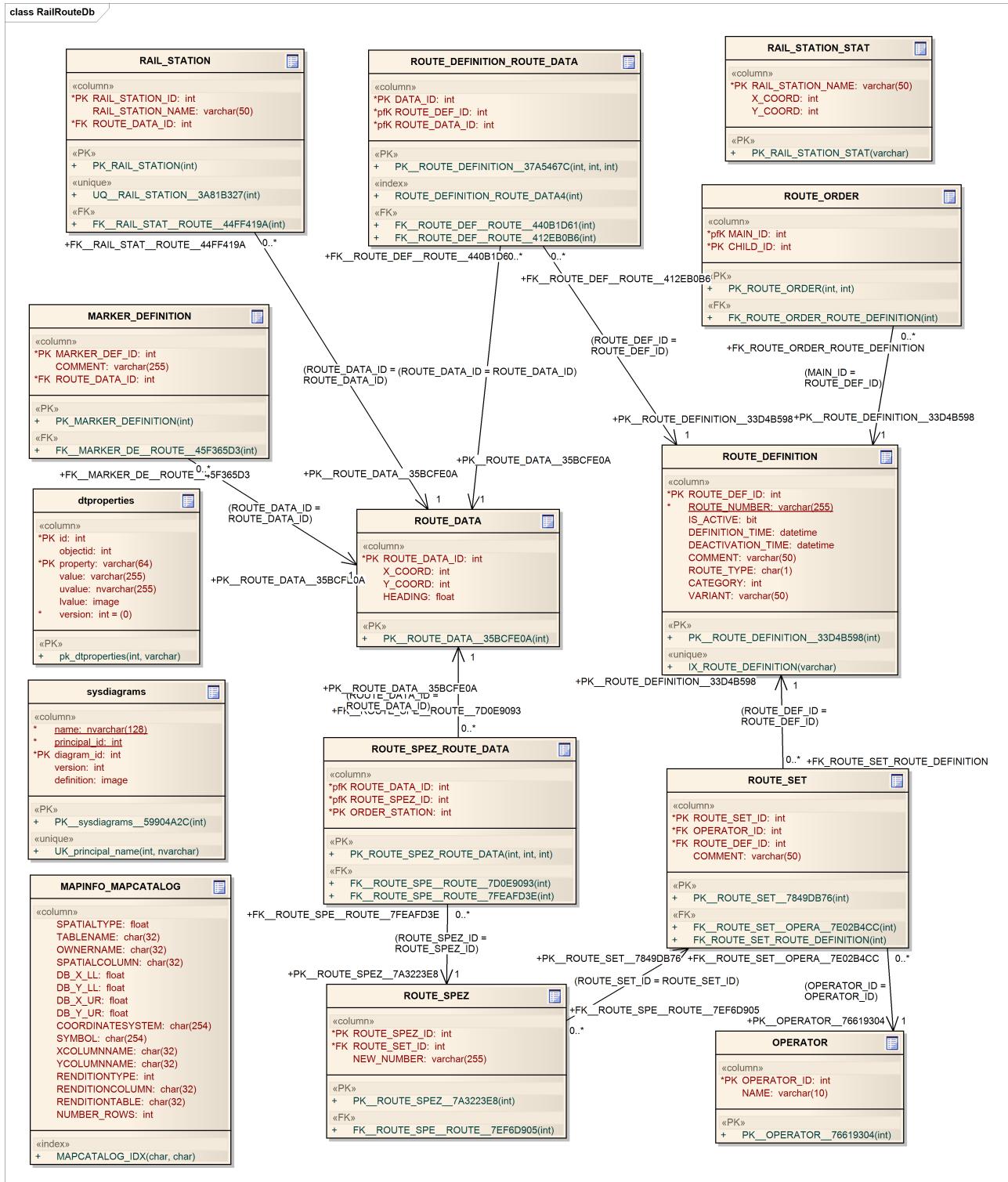


Illustration 20: Model of the existing RailRouteDb

1.16 Data relations

The following gives an overview of the data in the tables.

1.16.1 MARKER_DEFINITION

This contains the event markers.

There is some application redundancy in the marker and railway station table. Some stations, but not all are available as markers (in table MARKER_DEFINITION) and as railway station (in table RAIL_STATION).

1.16.2 RAIL_STATION

Relates railway station names and their geographic position.

1.16.3 RAIL_STATION_STAT

This seems to be legacy data. Here, in unnormalized form, the railway stations and their coordinates are available. Since, however this data is available via the RAIL_STATION table and its relations, this could probably be deprecated.

1.16.4 ROUTE_DATA

The geographic data is found in the table ROUTE_DATA. The markers and the Railway stations both are related to it via the ROUTE_DATA_ID key.

It contains 326009 items.

The coordinates are in the *Swissgrid* system, a national local coordinate system of Switzerland. The definition of x and y however is wrong with regard to the definition of the *swissgrid* system. In *swissgrid*, the x axis is pointing towards north, the y axis is pointing towards east⁸. Using the data requires to take this into account.

The heading information is in radians, ranging from -Pi to +Pi.

Given that there are 400 different routes (also called track segments) in the database, a single route has 800 waypoints. This seems to be handleable very well.

1.16.5 ROUTE_DEFINITION

This is the most complex table. It contains all the routes available. The routes are identified uniquely by the ROUTE_NUMBER, which is not the primary key of the table. It is however how the user will select the track segments, because this corresponds to the track segment number in the user interface.

For some track segments, there is a comment available, probably this could also be presented in the GUI.

The column ROUTE_TYPE contains a character that specifies, whether the route is a complete route, or whether it is a sub-route. Sub routes here are routes, that have a dot in their track segment number. They are part of larger routes.

The CATEGORY has no clear meaning, and could be deprecated.

The VARIANT seems to be an obsolete copy of the comment, and could also be deprecated.

⁸ See <http://de.wikipedia.org/wiki/CH1903#Kartenprojektion>

1.16.6 ROUTE_DEFINITION_ROUTE_DATA

This relates the waypoints and the track segments with each other.

The order of the waypoints is most probably given by DATA_ID (which strangely is also the primary key of that table).

It contains 530449 items.

1.16.7 ROUTE_ORDER

This seems to be an order for the routes. It seems that this could be neglected. Probably this is some custom order of the routes for presentation in the GUI. If necessary, this could get added in a later version.

1.16.8 ROUTE_SET

This relates routes and operators. The structure of this table is somewhat strange, because the column OPERATOR_ID contains all 1's, and the operator name is stored in text in the COMMENT column.

This table is however not relevant for the thesis. The same applies to the related OPERATOR table.

1.16.9 ROUTE_SPEZ

This assigns new numbers to the ROUTE_SET_ID. This is not relevant to the thesis.

1.16.10 ROUTE_SPEZ_ROUTE_DATA

This seems to assign railway stations in a specific order to the spez routes.

This is not relevant to the thesis.

1.16.11 MAPINFO.MAPINFO_MAPCATALOG

These are MapInfo specific definitions of tables. MapInfo is a mapping software and not part of this thesis.

1.17 **Tables to use**

The following tables are to be used for the thesis:

- ROUTE_DATA
- ROUTE_DEFINITION_ROUTE_DATA
- ROUTE_DEFINITION
- MARKER_DEFINITION

1.18 **Database Access**

The information about Tracks, Railway segment numbers, and Event Markers is found in the existing database. I have analyzed what data the Gui and Engine process need to access in which sequence and what data is to be transferred between the two.

To keep dependencies small, only the engine has access to the database. The GUI retrieves needed data via the engine, which in turn gets them from the database if necessary.

Data items that are exchanged

- A list of all available Track segments, by number. The data type here will be a list of strings, because the format is not strictly numerical.
- The available event markers for a given track segment. The data type here will also be a list of objects, of a custom event marker class.
- Track waypoints, for an area or for a given track segment. The data type here will be a list of objects, of a custom waypoint class.
- The user-selected track segment. The data type will be a string.
- A fired event marker. The data type will be a customized class, most preferably directly based on a corresponding class that is used inside the engine for position processing.
- Status data. This is only mentioned for completeness here. This data is completely unrelated to the database and thus not specified further here.

The diagram below shows the sequences for various activities related to the data exchange

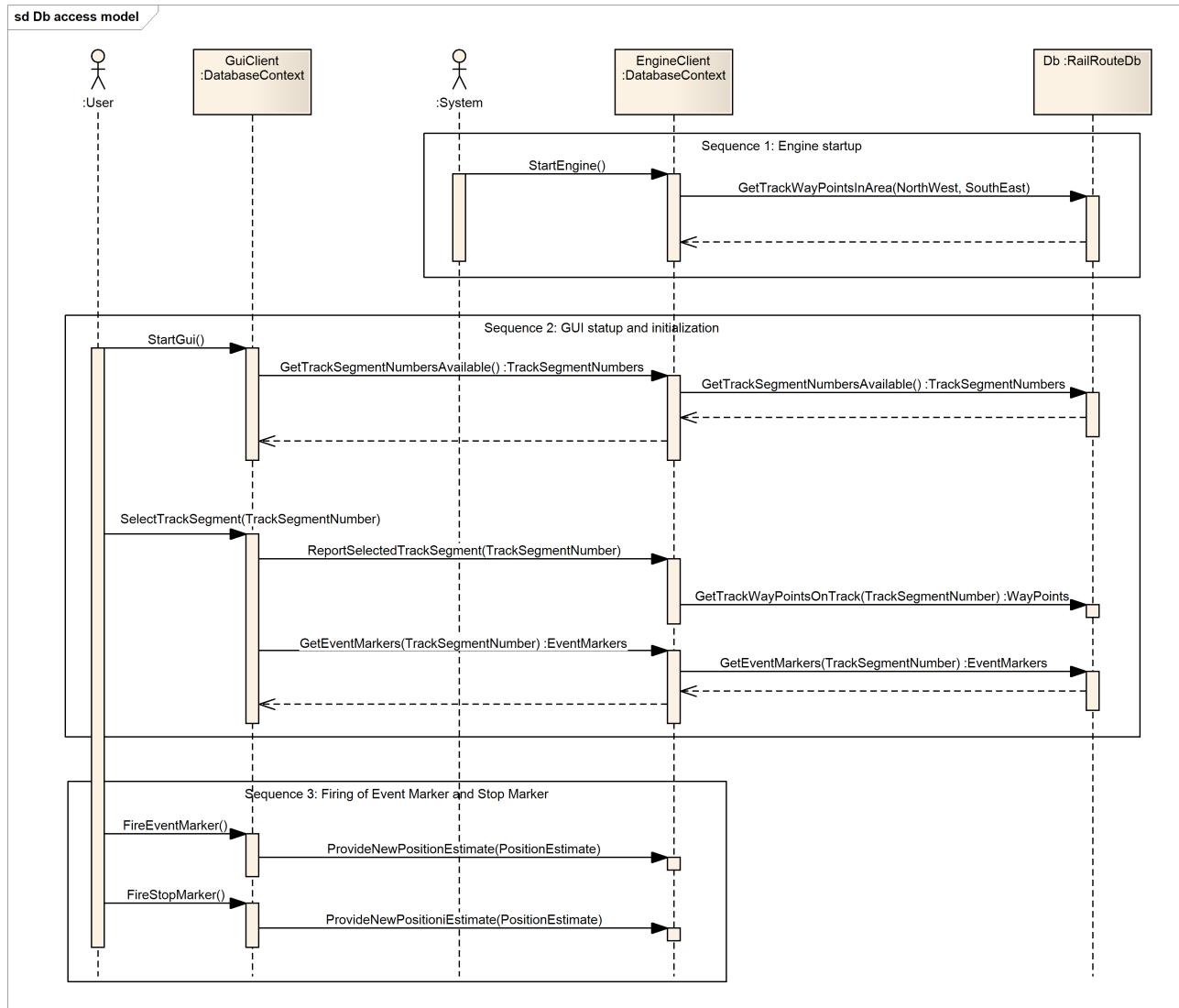


Illustration 21: Sequence diagram for the database access and data exchange

Sequence 1

This shows the startup of the engine. In this case, the engine gets track information from the database for an area around the current position, and the track alignment aligns to the nearest point on any segment within the area.

Sequence 2

This shows the start of the GUI. The GUI process first populates a list with all available track segments by their number. The user now may select the current track segment number. Then, after selection, the GUI client reports the selection, to allow the track alignment feature to start its work with using the track waypoints for this segment.

Also, the GUI client retrieves the event markers for the segment, to present them to the user. The user may then fire them afterwards.

Sequence 3

This shows the firing of an event marker and a stop marker.

1.19 Initial position

At startup, the system does not know its position from any sensor. To be immediately responsive, it should take the position estimate from the previous runtime.

1.20 Accuracy and Drift of the MTi-G INS

The MTi-G is the evaluated INS for the project. It has some configuration possibilities and has been tested in the office environment before use.



Illustration 22: The MTi-G evaluation kit for this thesis

The sensor offers a graphical visualization of its state and output with the accompanying software *MT Manager*.

1.20.1 Accuracy

In a static environment in the office, I have evaluated the accuracy of the sensor, in various GPS availabilities.

Therefore, the sensor was statically placed on the desk, while the GPS antenna was brought outside the window. The goal was to determine, what is the average CEP. The EKF Scenario used was "Automotive". See the manual for an explanation of those scenarios.

Good GPS reception

Analysis and Synthesis

The GPS antenna was placed outside the office window.

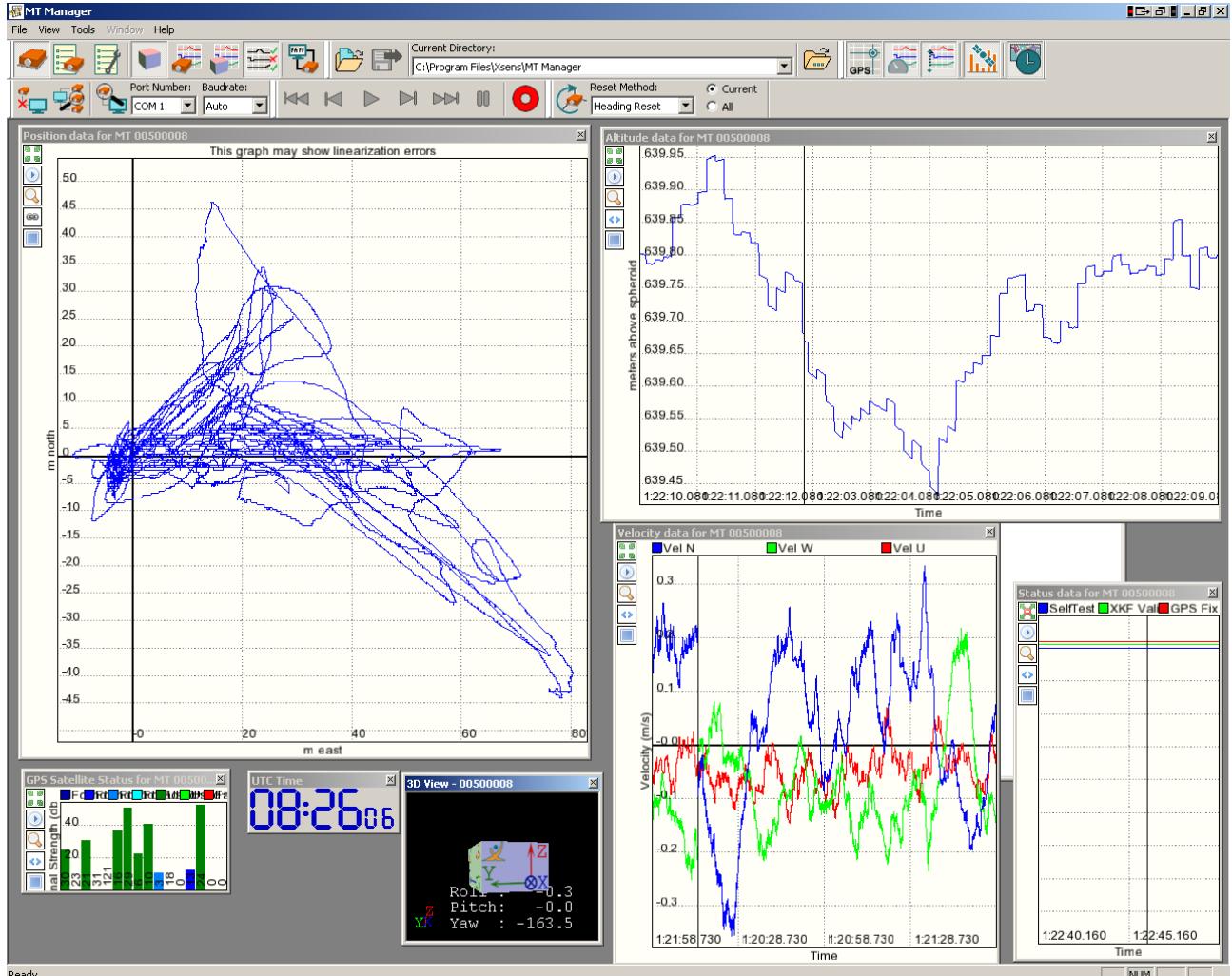


Illustration 23: MT Manager display with data from a non-moving sensor and good GPS running for an hour. The Illustration 23 shows that the CEP is about 10 meters, with some large, but seldom, outliers. The number of satellites used (green) 6 most of the time.

Bad GPS reception

The GPS antenna was placed inside the office window, after it had a good fix beforehand. This corresponds approximately the real situation in IC2000 cars.

The satellites used were 4 most of the time, but sometimes also no satellite was used. The signal level of the used satellites was around 20 dB less than in the good condition.

In situations, where the GPS is not available a slight drift occurs, but it is not significant as long as the GPS dropouts are short.

The CEP is larger as before, about 25 meters.

Analysis and Synthesis

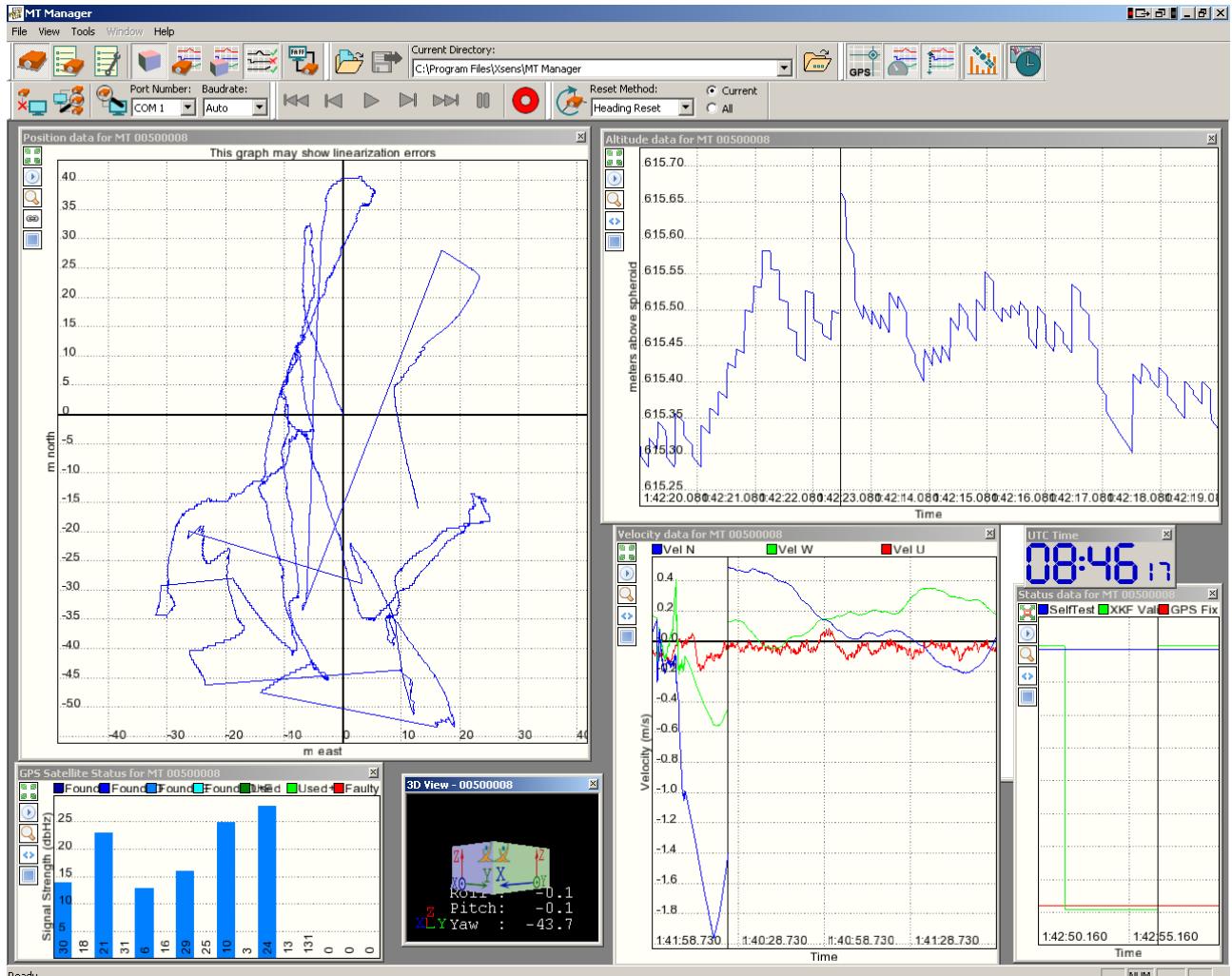


Illustration 24: MT Manager display with data from a non-moving sensor and bad GPS running for about 20 minutes

1.20.2 Drift

Drift was measured with no GPS reception. The GPS antenna was removed, after the receiver had a good fix beforehand. This corresponds approximately the situation when entering a tunnel with a good fix, except that the speed would of course be non-zero. The EKF scenario used was "Automotive".

A drift occurs, but this is much less than the specified 350 meter per minute. In the first try, I experienced a drift of 150 meters after 20 minutes.

In a second try the drift was 50 meters in 8 minutes.

Analysis and Synthesis

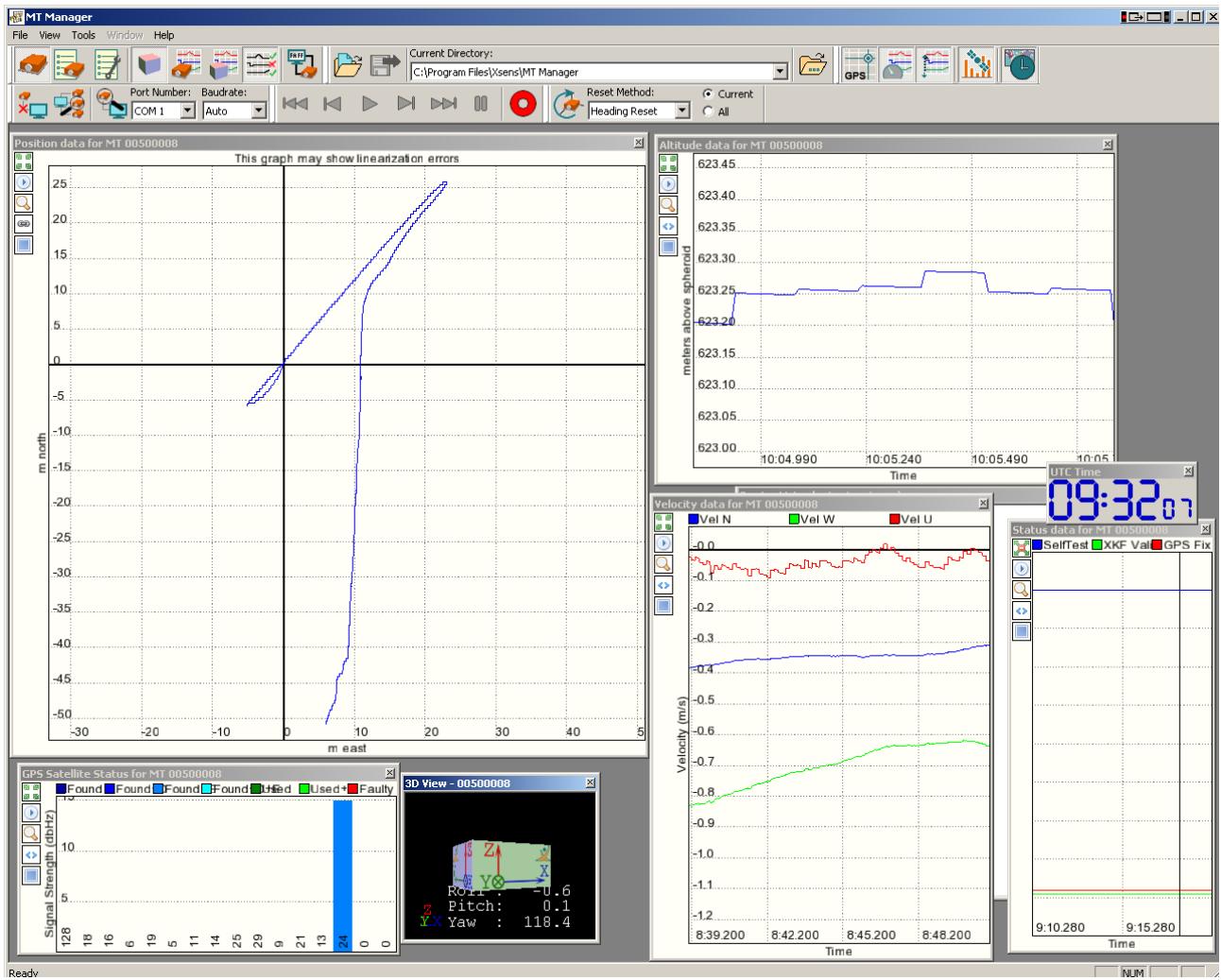


Illustration 25: MT Manager display with data from a non-moving sensor and no GPS running for about 8 minutes

Driving with a pushcart

To evaluate the behavior while moving, I have installed the sensor on a small pushcart, over the non-steering axis with the x direction pointing to the moving direction. Unfortunately, no clear position change was viewable in the position data window of the MT Manager. The position changed, but the trend did not correspond to the actual movement direction. Moving speeds were within the walking speed range.

Driving with a car

To further evaluate the unclear behavior observed with the pushcart, I brought the sensor into a car. The result stayed the same with more clearness. While the GPS antenna was connected, the movement of the car was visible very clearly. Once the GPS antenna is detached, the motion plotted corresponds to the real motion for a little longer than 10 seconds, then the sensor's position does not move further but stays at its place. When the GPS antenna is reattached, the position jumps to the right place again.

A support engineer from XSens clarified on this, mentioning that this behavior is by design. The goal of the MTi-G was to provide dynamic motion not absolute positions in the world. The sensor would rely "heavily" on GPS for absolute positioning.

This renders the MTi-G almost useless for the Thesis.

I now have decided to simply use the acceleration along one axis and integrate that into a traveled distance. The downside to this will however be that small amounts of misalignments in the vertical axis, as will often occur in real-world-scenarios, will lead to great misplacements over relatively short time.

Outside of this thesis, this problem may get fixed by using more advanced mathematical algorithms for tracking the position.

1.21 Integrating the Acceleration on a single axis

The MTi-G provides acceleration data in three dimensions. However, for the thesis, for simplicity, only the acceleration measurements on a single axis is taken. This axis should be very well in line with the direction of movement of the train car and be kept horizontal.

From the periodically measured acceleration, the current position can be deducted using the following formulas.

They base on numerical integration, using the *midpoint method*⁹. This method is more precise than the traditional *euler method*¹⁰.

The mean average between the current and the last acceleration measurement

$$a_{mean} = \frac{a_{t-1} + a_t}{2}$$

The current velocity, where dt is the time difference between the last and the current acceleration measurement:

$$v_t = v_{t-1} + (a_{mean} * dt)$$

The mean average between the current and the last velocity calculation:

$$v_{mean} = \frac{v_{t-1} + v_t}{2}$$

The current position (r_t) where (r_{t-1}) is the position at the last measurement.

$$r_t = r_{t-1} + (v_{mean} * dt)$$

9 See http://en.wikipedia.org/wiki/Midpoint_method

10 See http://en.wikipedia.org/wiki/Euler%27s_method

2 Design

2.1 Package Diagram

2.1.1 Diagram

The package diagram honors the requirement of the separation of the GUI part and the Engine part. To allow data transfer between the two processes, an exchange package is created. Illustration 31 shows the packages with the most important dependencies.

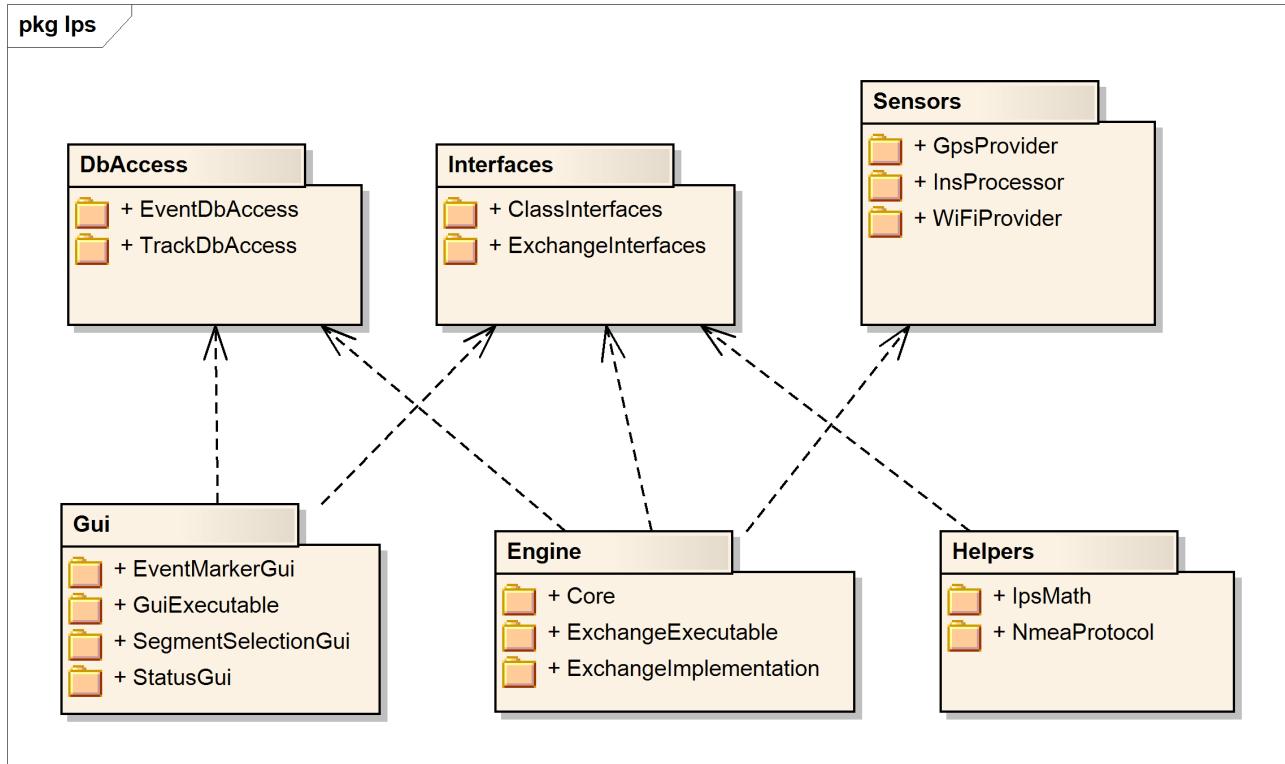


Illustration 26: High-Level Package diagram

2.1.2 Package description

DbAccess

This allows the engine process the needed access to the track- and the Event-Database. The access to the WiFi database is encapsulated entirely in the sensor package, because this is not needed anywhere else.

Interfaces

This package contains all the interfaces needed in the whole project. Therefore every package may depend on it, although for simplicity, only the main dependencies are shown.

The exchange interfaces are used for data transfer between the Gui and the engine process. This will include event marker retrieval and position provisions by the event marker firing in the GUI.

To prevent circular dependencies, this package must not depend on any other package in this system. It will depend of course on the .NET library.

Sensors

This package encapsulates all positional inputs to the system, except the user-fired event markers.

Gui

This package contains all parts of the GUI, like the forms, dialogs etc. It needs database access, to allow the user the input and verification of the track segment number and to fire the event markers.

Engine

Contains the processing of position estimates, the output of data to the serial port and provides access to all necessary data via the exchange interface.

Helpers

The helper package contains various classes, especially such for coordinate conversions and calculations plus parsers for the NMEA protocol. Every other package may depend on it, but for simplicity, this is not shown.

2.2 Engine core

The core engine's elements should be replaceable, therefore, I have chosen to extensively use an interface based design approach.

2.2.1 Business class diagram

Diagram Illustration 27 shows the business class model for the engine core. The classes for database access, inter-process communication and the GUI have been omitted intentionally, to avoid unnecessary complexity.

Design

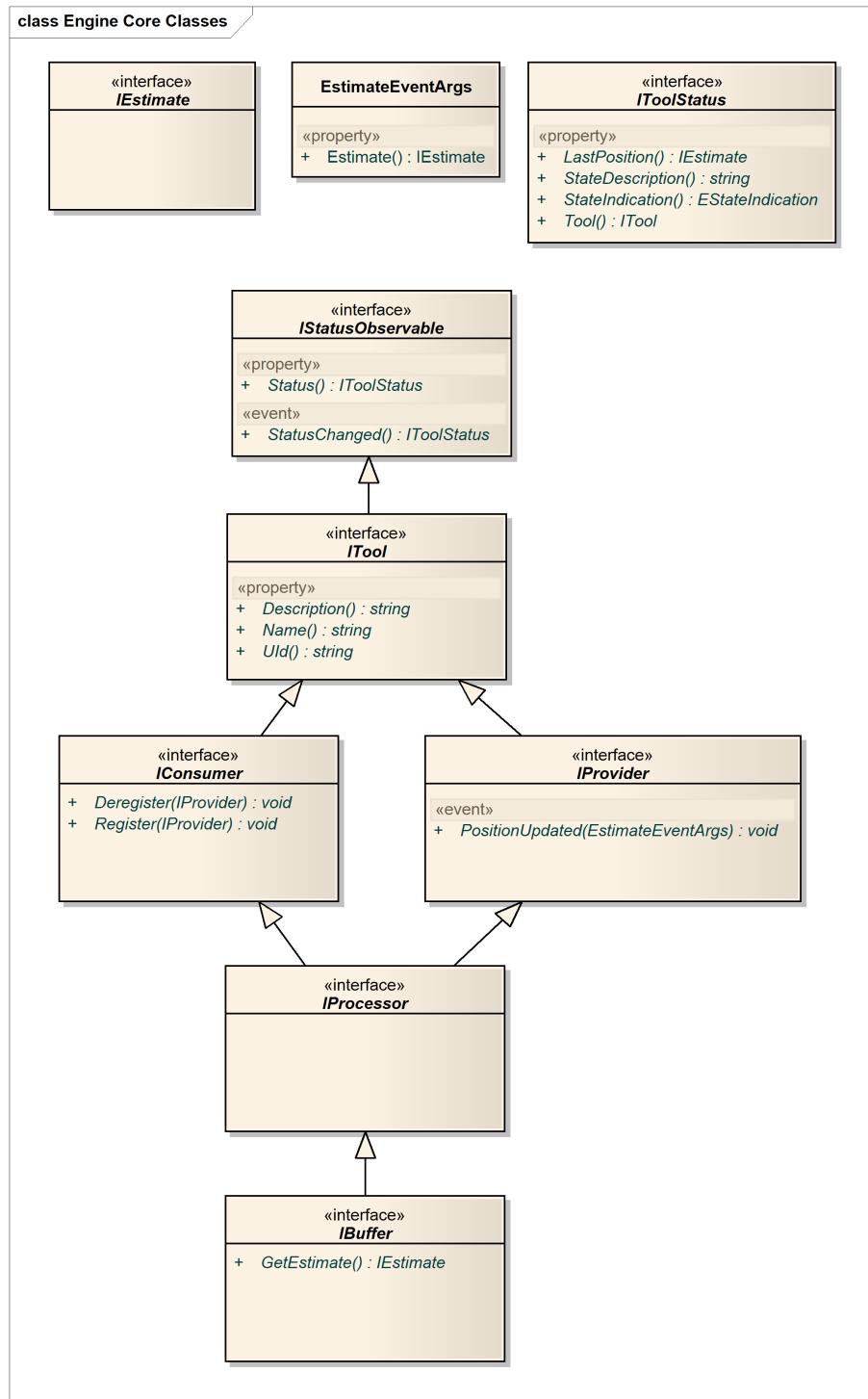


Illustration 27: Business class model for the position estimate processing

The classes are all about position estimate processing. **IProvider** and **IConsumer** classes could be thought of as observable tools. They represent just various levels of features for accepting, processing and providing of estimates. These tools provide also status information, defined via **IObservable**.

An **IConsumer** implementation should keep the references that it has attached to, for later usage. This is especially necessary when it wants to query an **IBuffer** intermediately.

Design

ITool defines identification for a tool. This will later simplify the GUI code for status presentation and allows for easy differentiation of tools in code.

An IBuffer does all the same as a processor, but has also an internal storage. It can either be used as static buffer for a single estimate or implement an extrapolation algorithm. If an IConsumer is attached to an IBuffer, and wants to intermediately access the buffer (instead of just waiting for the Available event), it must keep an internal reference to the IBuffer object, so it can access the getter. The GetEstimate() method. could also get implemented as read-only property.

An IPProcessor will be used for most of tasks: They could serve as mixers, as the Aligner and so on.

An IProvider will implement the position providers using the various sensors.

Current Best Position Estimate

Design

A singleton, called “Current Best Position Estimate” will hold the best estimate at any time. It could be implemented as Singleton of an IBuffer at the very end of the tool chain. Anyone interested in this information could just statically query the singleton.

2.2.2 Example Object diagram

To prove the business classes, I have created an object diagram for concrete case of this project, with all inputs and the NMEA output. See Illustration 28.

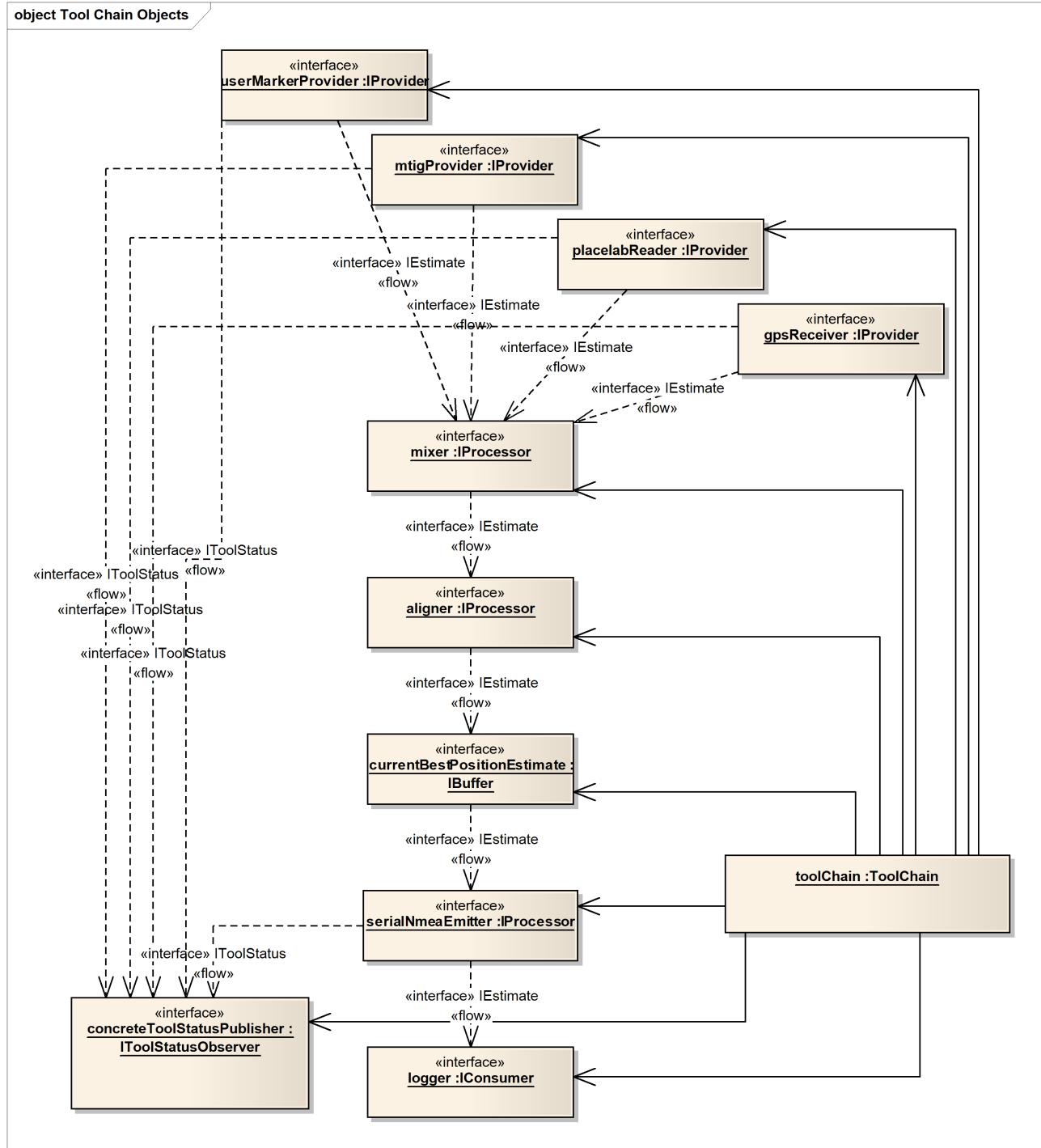


Illustration 28: Object diagram of the engine core

The mtigProvider object will implement the error correction algorithms discussed earlier.

The toolChain is a singleton instance of the ToolChain class and creates and wires all objects.

According to the requirements, status information for the providers and for the serial NMEA emitter is transferred back to the GUI, via concreteToolStatusPublisher.

The serialNmeaEmitter is an IProcessor, to simplify logging of its output. The emitted position could be provided as its Available() event. The logger is the an IConsumer that simply logs the provided estimate.

The diagram shows the fitness for the design to provide more logging and to attach also more tools to the the status publisher. In general, the whole system is easily rearrangeable by changing the implementation in the ToolChain singleton instance.

2.2.3 Sequence diagram for building the chain

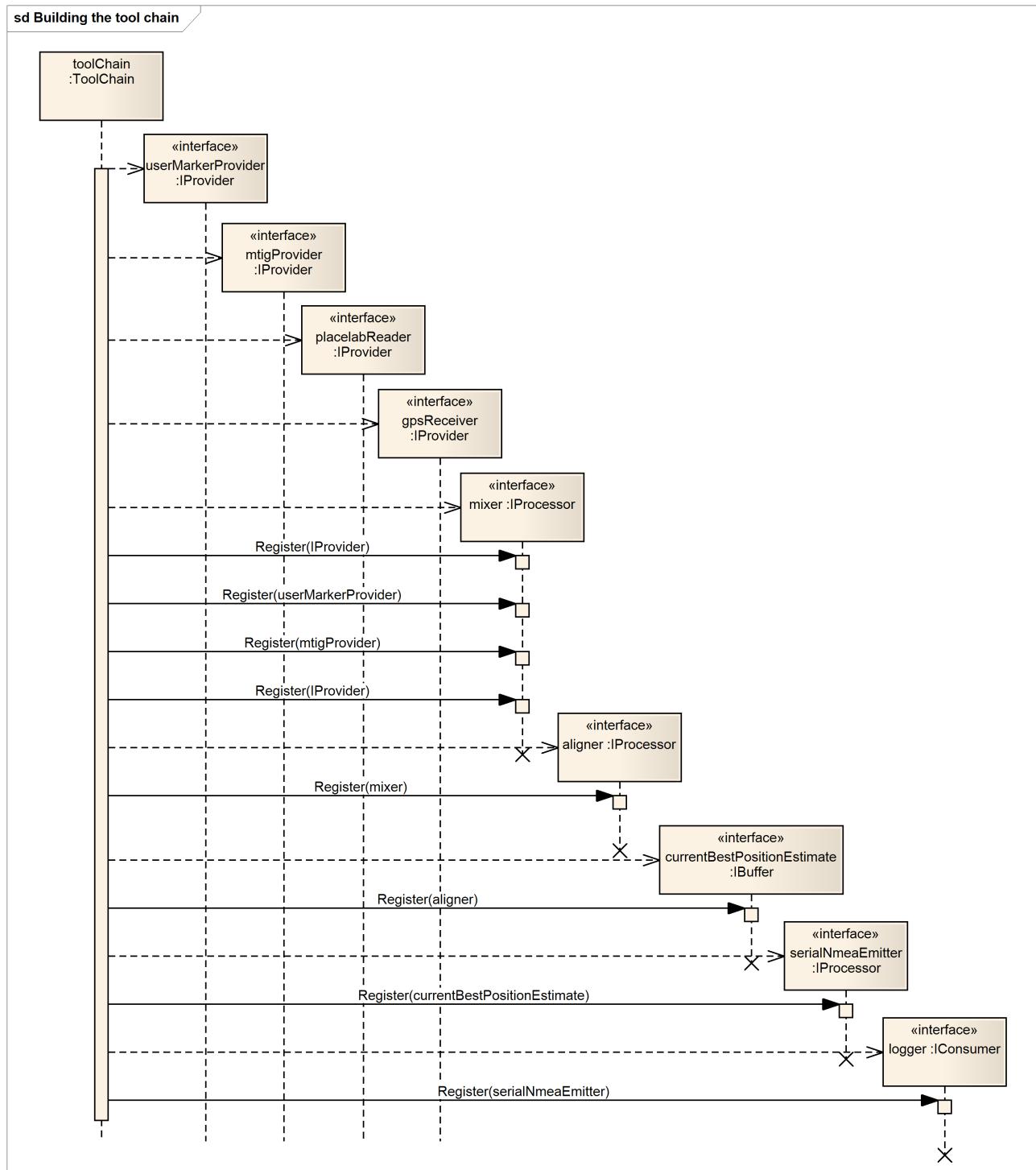
For further illustration, I have drawn a sequence diagram that shows how the tool chain is built up. As example serves the object diagram from Illustration 28.

Building the chain is rather simple. The tool chain factory creates the tool objects and wires them to a chain of tools. The factory calls an IConsumer's AttachTo() method, with the preceding IProvider as argument. The IConsumer then registers on the IProvider's Available event. It thus will get notified each time the IProvider has a new estimate.

The order of attaching the objects is not relevant.

With the same mechanism, the factory can also initiate a deregistration, by calling the DetachFrom() method.

Design



2.3 Data Exchange

2.3.1 Using WCF

To keep the data exchange simple, the design relies on Microsoft's Windows Communication Foundation (WCF). This involves the design of ServiceContracts and DataContracts.

Design

Track Segment and estimate data exchange

The user will input the track segment number. Illustration 30 shows the design for this.

Also, this diagram shows the submitting of position estimates originating from Event Markers and train stop markers.

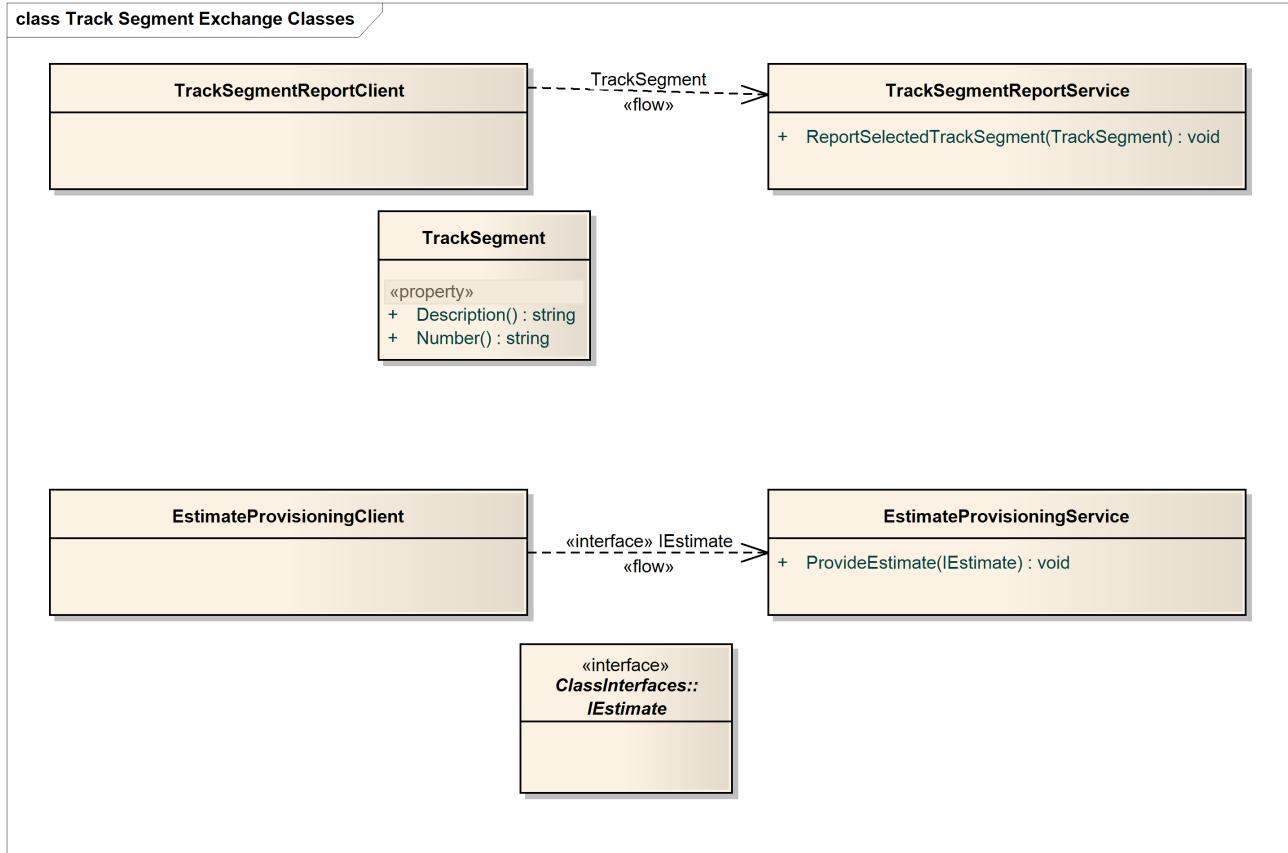


Illustration 30: Track Segment Exchange Classes

The **TrackSegmentReportService** will be instantiated as singleton and session-oriented in the WCF-sense. It will allow only one session to be active at a time, because only one user is allowed to input the track segment numbers. Allowing more than one would cause inconsistency. The **TrackSegment** Class will serve as DataContract for the data exchange. The **CheckUserAliveCallback** will serve as callback, for the alive-checking.

The **EstimateProvisioningService** will be instantiated as singleton, too. However, an arbitrary number of clients is allowed. Since the data of the clients is not influencing each other and there is no need for a session, the WCF-communication will be "per call".

Validation of track segment number

In case the GUI process has crashed or the user is not responding to the system anymore, the corrected error in the alignment processing unit may grow to large number, when the system is not on the last selected track anymore. Therefore, the alignment processor should validate the currently track segment number and discard it in case it seems to be wrong. The criterion may be that the correction is larger than a factor of the maximum expected error of the system under good conditions. The implementation may choose this value upon experience. Another criterion should be if the GUI is knowingly terminated. This get detected by the ending of the session with the GUI.

Status data exchange

Design

The status data exchange is solved with the publisher/subscriber pattern and the observer pattern (with events instead of method calls). Illustration 30 shows the design classes for this.

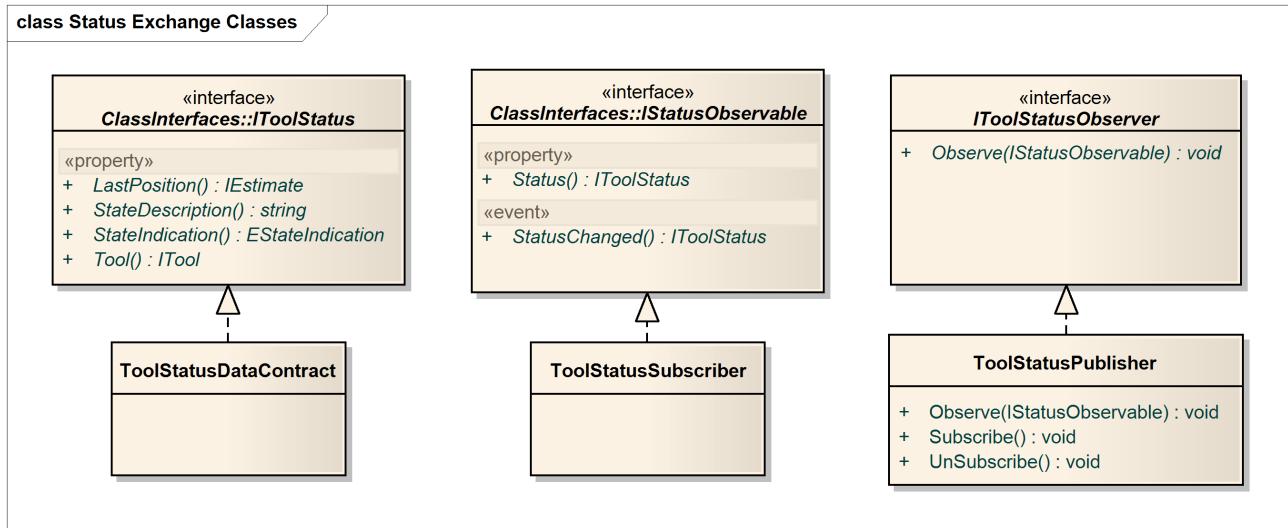


Illustration 31: Status Exchange Classes

The `ToolStatusPublisher` is the class maintaining the `ServiceContract`. It observes one or more tools (attached via the `Observe()` method) and publishes its data to all subscribers. Subscribers can subscribe and unsubscribe to the service via the respective methods. Subscription and publishing back to the subscriber will involve a callback, which is not shown at this stage. The publisher is hosted with the instance context mode set to "Single", because it must present the same information to all subscribing clients.

The data contract is the `ToolStatusDataContract` class.

Defining the status observation and the publication of the information works similar as wiring the chain. The `toolChain` singleton creates the publisher, and initiates observation of the relevant tools. The sequence diagram in Illustration 32 shows the process, with the object diagram in Illustration 28 as example again, where only the providers and the emitter is observed. The creation of the tools is not shown.

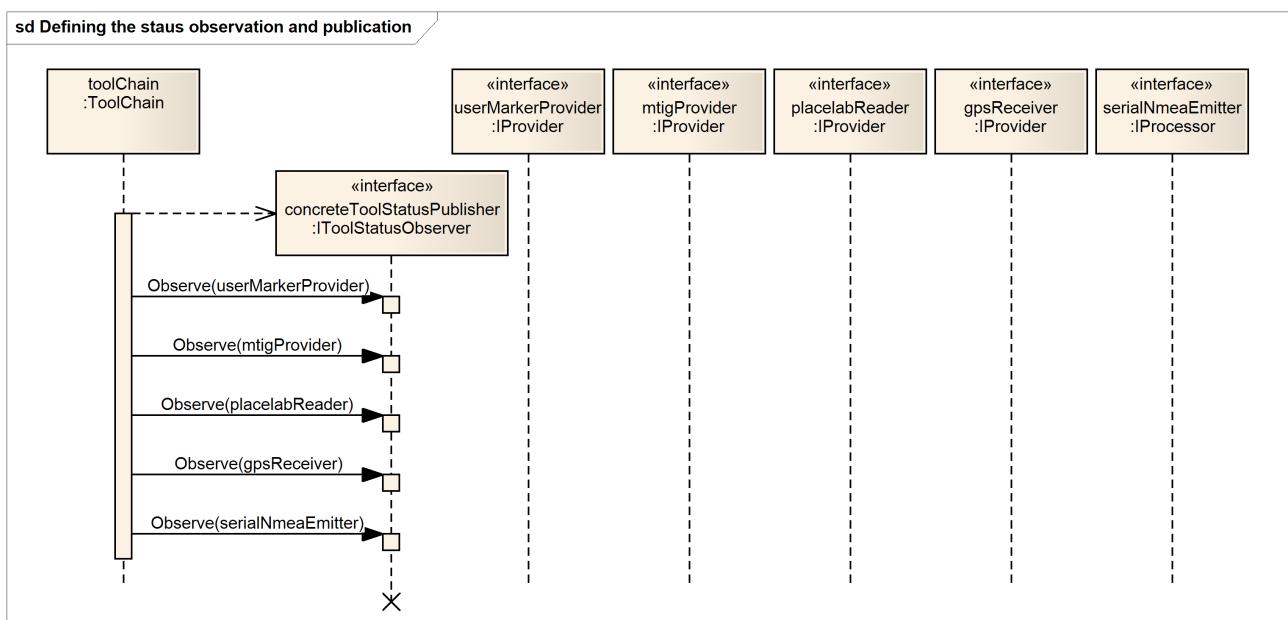


Illustration 32: Defining status observation and publication

3 Implementation

3.1 Overview

The solution is divided in 8 Packages containing a total of 123 classes and interfaces. In the following some important aspects are presented. The code is fully documented within the source files.

The source is not public, but may be requested from the author.

3.2 Changed packaging

During the implementation, the packaging has been slightly adopted. The namespace convention is Inventis.Ips.{PackageName}.{SubpackageName}

Packaging is now as follows:

Implementation

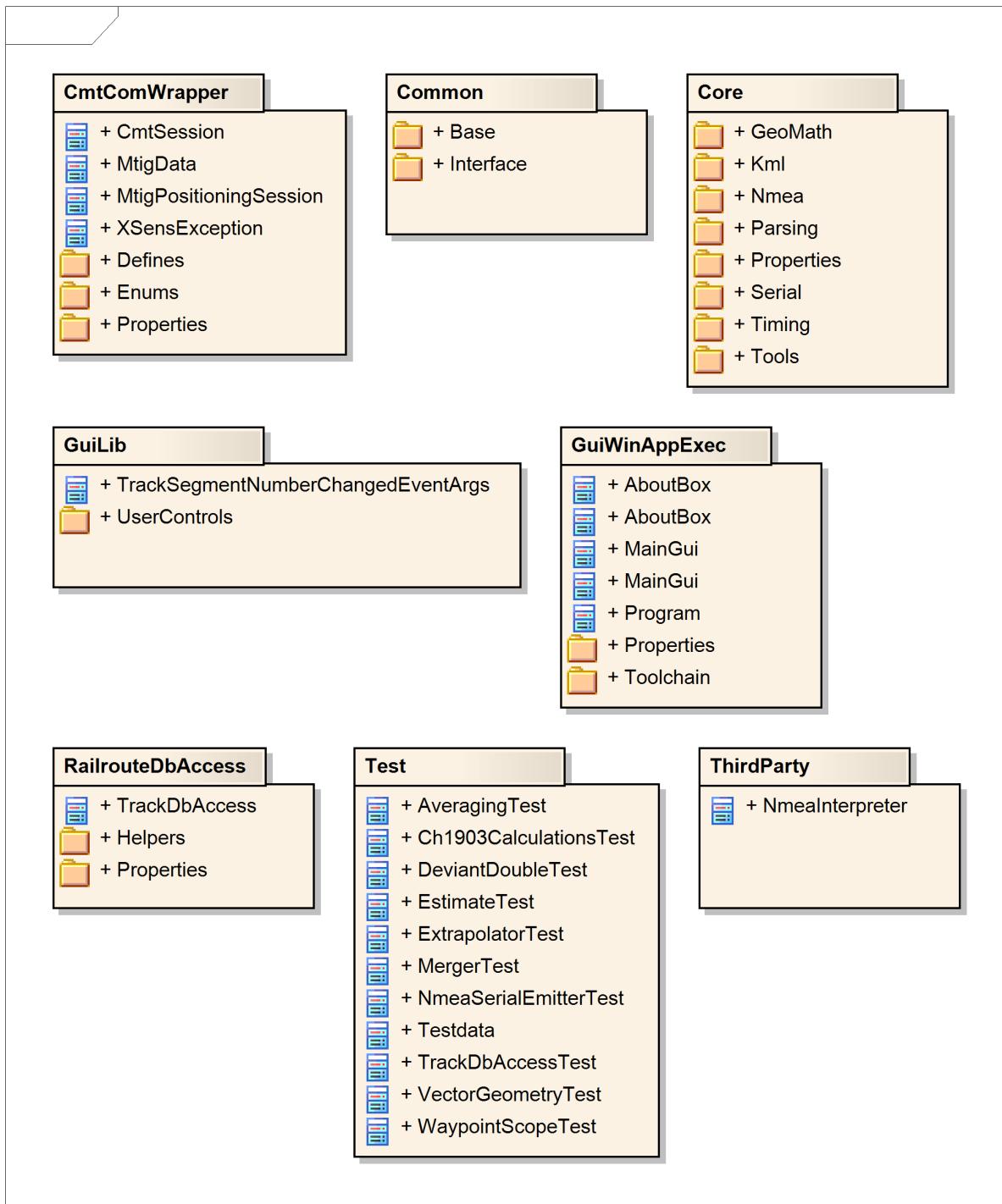


Illustration 33: Implemented Packaging

3.2.1 GuiWinAppExec

The GUI executable, containing both the engine and the GUI part. The classes for building the tool chain are also contained here.

3.2.2 Test

Unit Test classes for the whole solution

3.2.3 Common

Components that are used in the whole solution

Common.Interfaces

Interfaces for all sorts of components, including the ones for data exchange between GUI and engine.

Common.Base

Base implementations where the definition of Interfaces is not sufficient (example: EventArgs classes)

3.2.4 Core

The core of the engine. It contains all classes for the sensors, the tools with their processing, logging and the data output.

Core.Tools

Implementations of the various tools.

3.2.5 RailrouteDbAccess

All classes for the access to the track database. An Interface is provided to make this package exchangeable.

3.2.6 GuiLib

Controls and other items for creating the GUI

GuiLib.UserControls

UserControl-derived classes with specific functionality for the GUI part.

3.2.7 CmtComWrapper

A wrapper around the COM-DLL for the MTi-G sensor. It provides sessions and basic error handling.

3.3 *Calculating the heading deviation for GPS data*

The problem here is, that the NMEA data used does not provide an indication of the accuracy of the heading. It is a known problem for GPS receivers that the heading accuracy is low on low speeds. I take this partially into account, by calculating an assumed uncertainty by using the traveled distance and the CEP values of the positions.

I use a simplified approach by using the traveled distance since the last position update and I simply add the CEP of the last and the current position.

Angular error = $\text{atan}((\text{currentCEP} + \text{lastCEP})/\text{distance})$

3.4 Tools

The tools are a major part of the solution. There is an interface, ITool, and various derivations of it, be it other interfaces, abstract base classes and concrete, instantiatable classes.

3.4.1 Interfaces

The interfaces serve as joints for building the chain. They are all contained in the Common Package, in the interfaces sub package.

Compared to the design interface classes, the `Consume()` method has been added to the `IConsumer`, to provide a means to explicitly feed Estimates into the chain. This is used for the input of the event markers from the user.

3.4.2 Abstract base classes

Two abstract base classes are contained in the Base sub package of the Common Package, since every tool should derive from one of them to fulfill the requirements. More abstract base classes are defined in the Core Package, for implementation convenience.

The abstract base classes are providing base functionality for the implementation of the concrete classes. This saves code lines and makes changes of the common behavior more easy.

Implementation

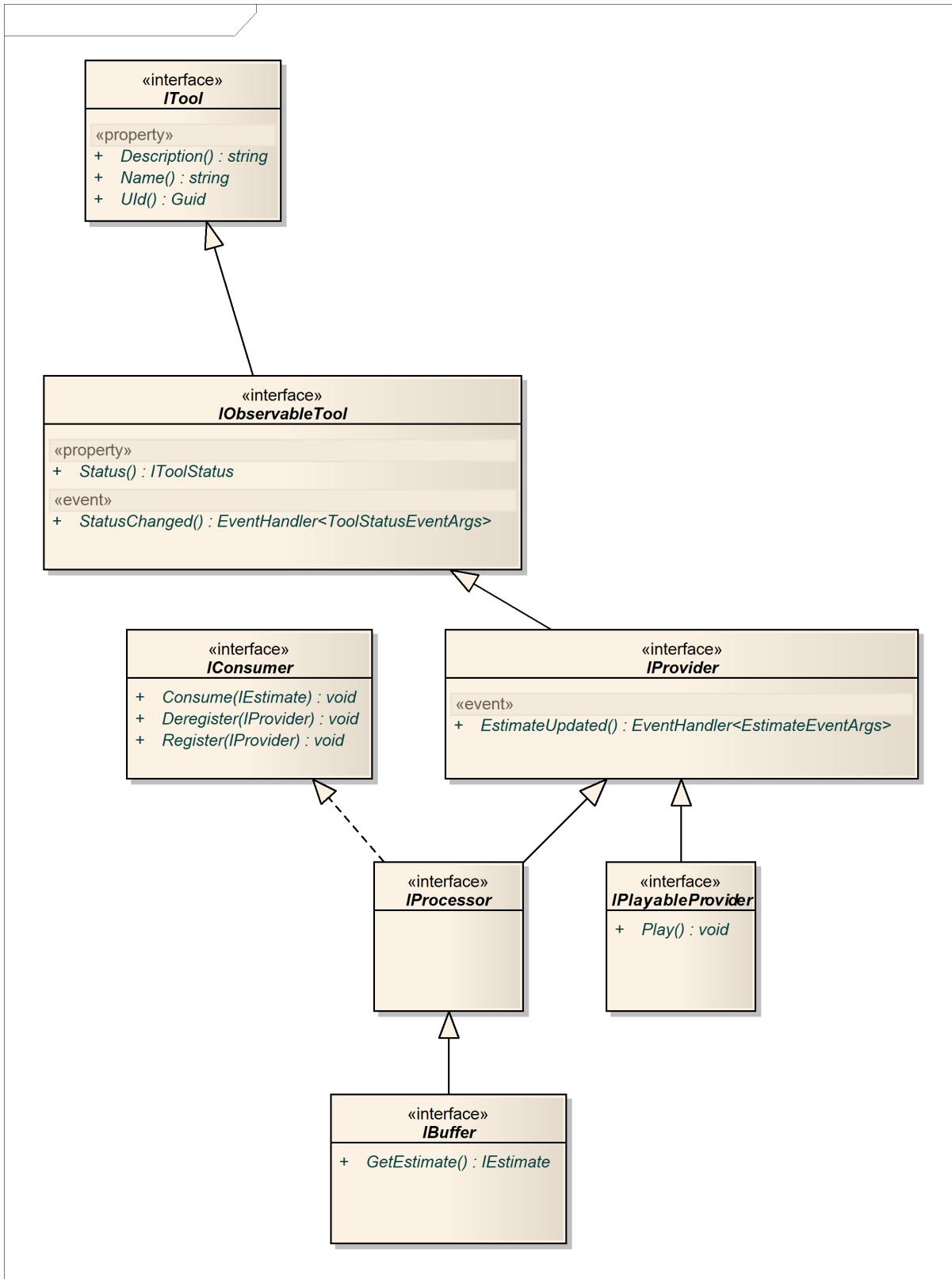


Illustration 34: Interfaces in the tool chain

Implementation

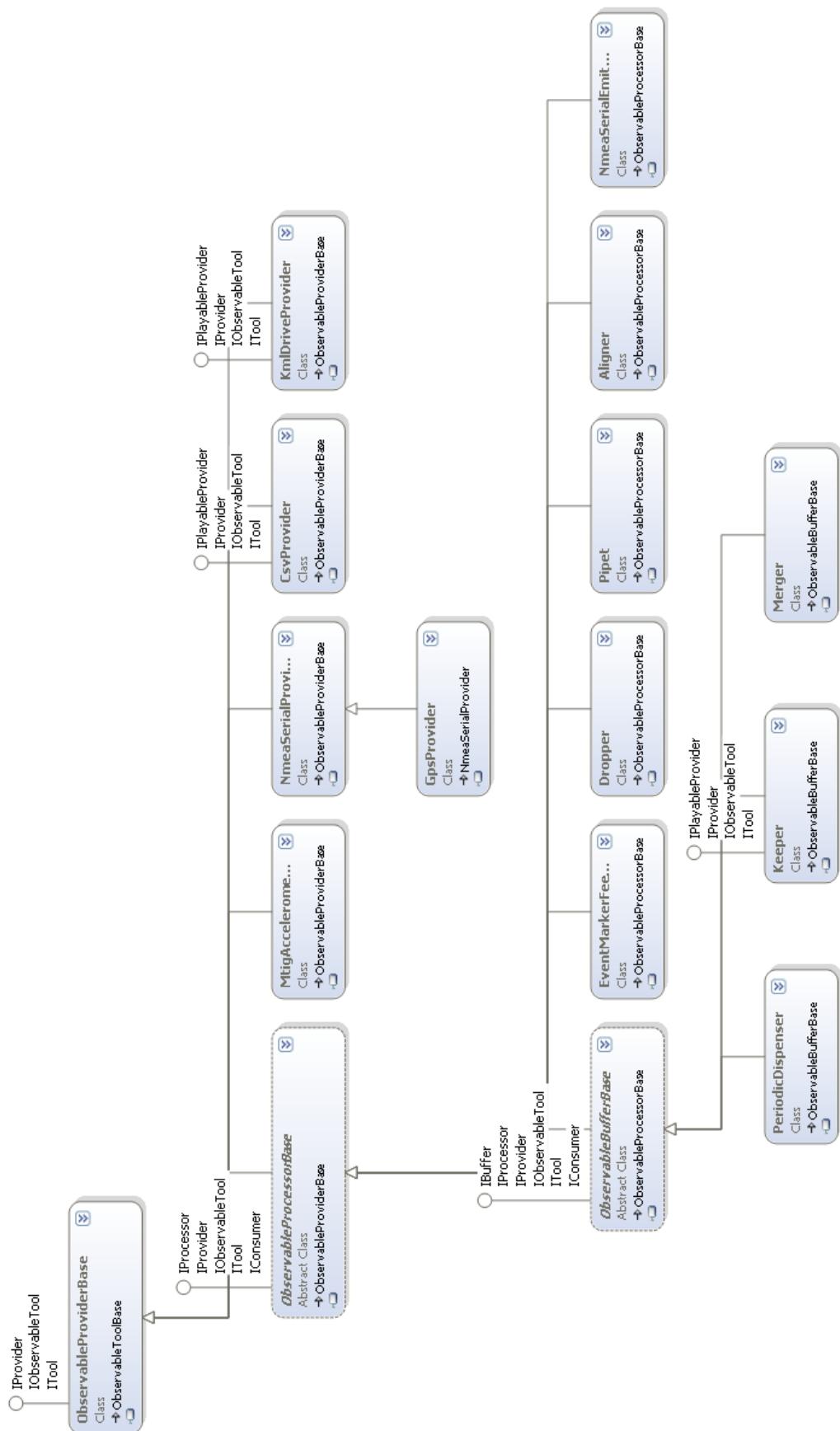


Illustration 35: Abstract and concrete tool classes, overview

3.5 Using virtual serial ports

Com0com is a tool that provides connected serial ports without real hardware, in Windows XP. It thus servers as a virtual null-modem cable. This is useful for testing purposes and is used to fulfill the requirement #57.

The configuration of the serial ports however, was not so easy. They should allow to send and receive data even when no device is connected. The following configuration worked:

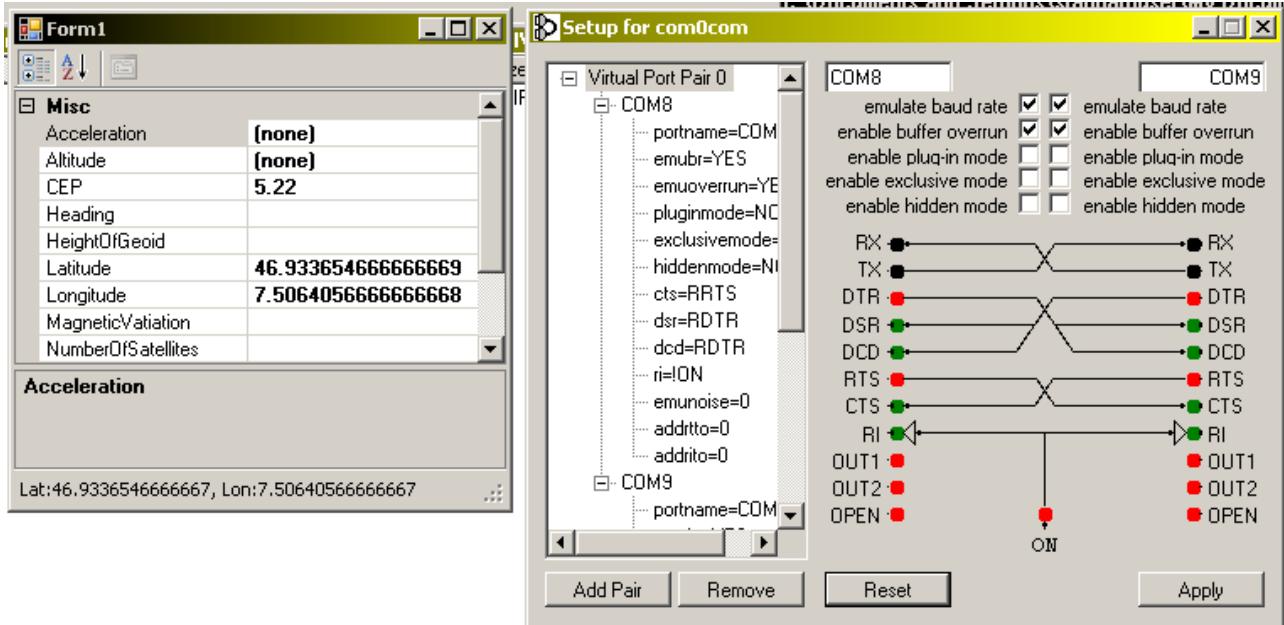


Illustration 36: com0com configuration for serial data output

3.6 Checking the CEP calculation for GPS

To see whether the formula for conversion between HDOP and CEP is correct, I have run a test, where the u-blox EVK-5H GPS receiver was stationary, and I observed the position estimate in Google Earth, using the Earth Bridge tool. I could be observed, that the CEP roughly coincides with the position shown in Google Earth, it was even a little pessimistic. The used specific accuracy of the GPS receiver was 6m.

Implementation

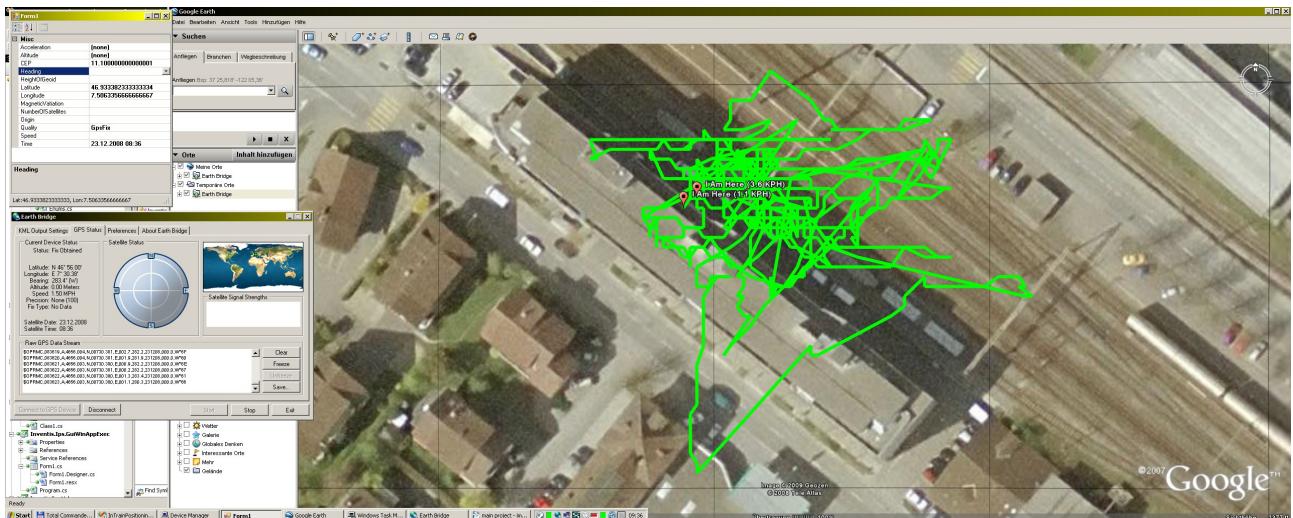


Illustration 37: Checking the implemented conversion from HDOP to CEP

It is also visible, that the resolution for the Latitude and Longitude values in the GPRMC Sentence should get increased.

3.7 Implementing the INS provider

3.7.1 Using the SDK from XSens

The SDK provides ready-to-compile examples for using a DLL for accessing the MTi-G, as well as other, more low-level examples.

P invoke

A first approach used the platform invoke feature of .NET, but the implementation of all the methods was quite tedious. This approach has been dropped in favor of using the COM object.

COM Object in the DLL

As I have found, the SDK provides a COM object in their DLL. Since it is very easy to write code against a COM object DLL in Visual Studio, I went this way. This will require installing the COM DLL on the target system. The SDK installer however is able to do this.

In visual Studio 2008, using a COM DLL is as easy as just adding a reference to it in the current project.

Setting the current position

The SDK documentation mentions a method for setting the position of the sensor in the the Latitude/Longitude coordinate system. I have tried to use this function, but there is an error returned. Probably this function is only intended for devices that do not have a GPS inside. It seems, that the statement of XSens, that setting the position of the MTi-G is not possible, is true.

3.7.2 Configuring the MTi-G

Various options are available. Refer to [XSNS08UM]

I have decided to use an update rate of 10Hz. This is the slowest available and sufficient for the intended update rate of the system's output of 1Hz. As orientation mode, I use the Euler angles. These are the most simple to understand angles. The singularity within is not a

problem, since the sensor will have a defined orientation in the train. The value I used as heading is the one called "Yaw".

3.8 Representing global coordinates

I decided to use 3D coordinates throughout the project, even when the track database does not have 3D data.

Some coordinate mangling is done via a 3rd-party library, *Gavghan.Geodesy*, which now also has a reference to the Common package. This does not introduce circular dependencies as long as the 3rd-party library does have no other dependencies in the solutions. The library contains ready-made types for representation of positions on the globe.

3.9 Implementation of the database access

For the database access, LINQ to SQL is used. Visual Studio 2008 has excellent support for creating access classes to a Microsoft SQL server database, a so-called O/R mapper. There is good support on the Internet for this feature.

Using this, the implementation of the access classes, the so-called database context, was made within minutes.

3.9.1 Interface

To allow defined access to the RailRouteDb, and to encapsulate the database internals, an interface for accessing the database was defined.

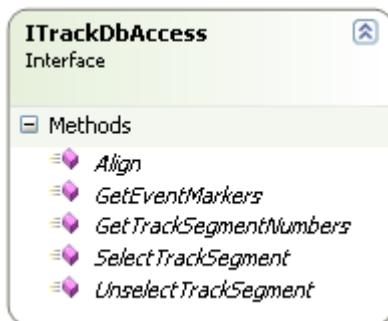


Illustration 38: Interface for accessing the track database

The first method to call should be the `SelectTrackSegment()`, because this will define the behavior of all subsequent calls to the other method. See the code documentation for more details.

3.9.2 Designing the database context

Visual Studio 2008 provides easy creation of a database access context by dragging and dropping of the relevant tables onto a design surface. The necessary code is then generated automatically. I used this feature with success.

Of the many tables of the database, only four were used, as shown below.

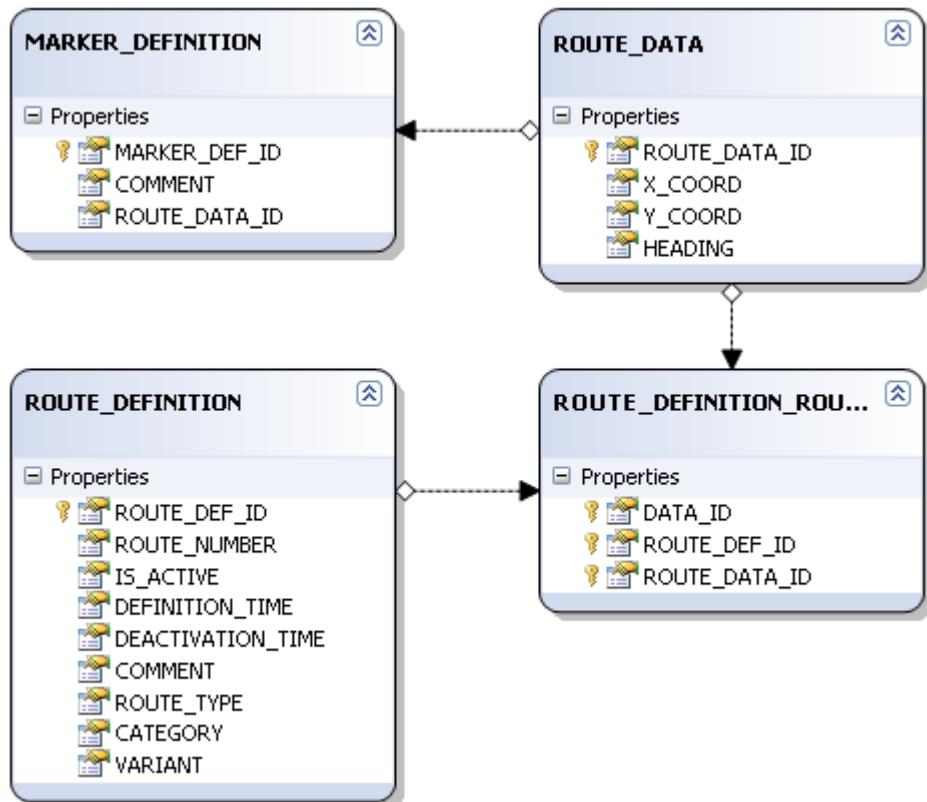


Illustration 39: Used tables for the track database access

3.9.3 Getting the track segment numbers

Getting the track segment numbers is done via a linq query that simply returns a list of strings of all ROUTE_NUMBER's in the ROUTE_DEFINITION table.

Implementation

Implementation

Implementation

3.9.4 Getting the nearest point on the tracks

The algorithm to find the nearest point on the tracks works using a small scope which is containing a set of points of a center point. This set acts as some kind of proxy to minimize the database queries. The scope is implemented in it's own class.

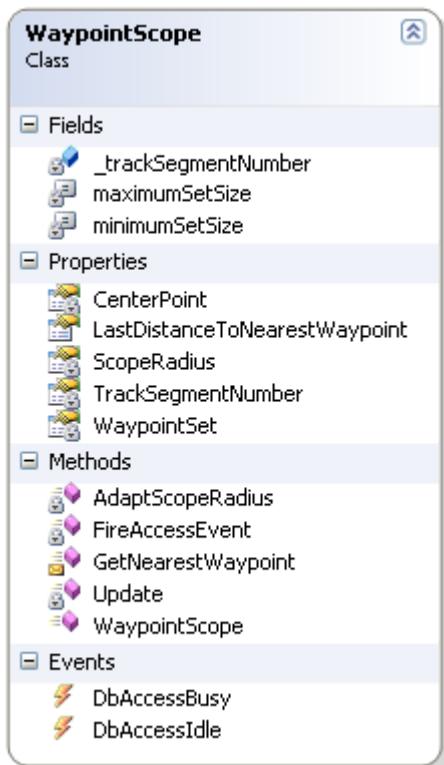


Illustration 40: The WaypointScope class

Since the data in the current RailRouteDb also has heading information for each point, the analyzed method of using segments between two or three consecutive waypoints was dismissed. The heading information is now used to create a line through the waypoint, on which the nearest point to the input position is calculated.

This algorithm has the advantage that it only needs a single waypoint and was therefore much easier to implement. The drawback is that it relies very much on the the heading data, where small errors could yield quite large errors. In practice this is the case when the track has much curves and the distance from the input estimate to the next waypoint on the track is large. Long-term tests with real data could tell how big this issue is. In other, more frequent cases, the results are of reasonable precision.

Simulation

To assess the performance of the algorithm, I have made a simulation using Google earth.

The data used consists of three parts

- The position output of the software, fed into *Google Earth* using *EarthBridge*. (shown as thick green line)
- A KML file of the track data of the track segment 231, starting in Bern, Casinoplatz and ending in Worb (shown as thin white line).
- The output of the "Driving Directions" feature in *Google Earth*, when driving from Bern to Spiez. (shown thick as violet line).

The simulation evaluates the various difficulties for the algorithm used.

At the starting point, at the Casinoplatz, the drive track circulates around the starting point of the track. It is clearly visible how the drive track gets tied to the extension of the track.



Illustration 41: At the Casinoplatz

Implementation

When the track database has many close waypoints, the tyeing is very well, as for example visible at the Thunplatz.



Illustration 42: At the Thunplatz

Implementation

Also at large correction distances, the algorithm works well, as the example from the Ostring shows. The car drove away from the track to the highway, but the tied position remained on the track all the time. When the track distance growed fast, the tied points made jumps two times.

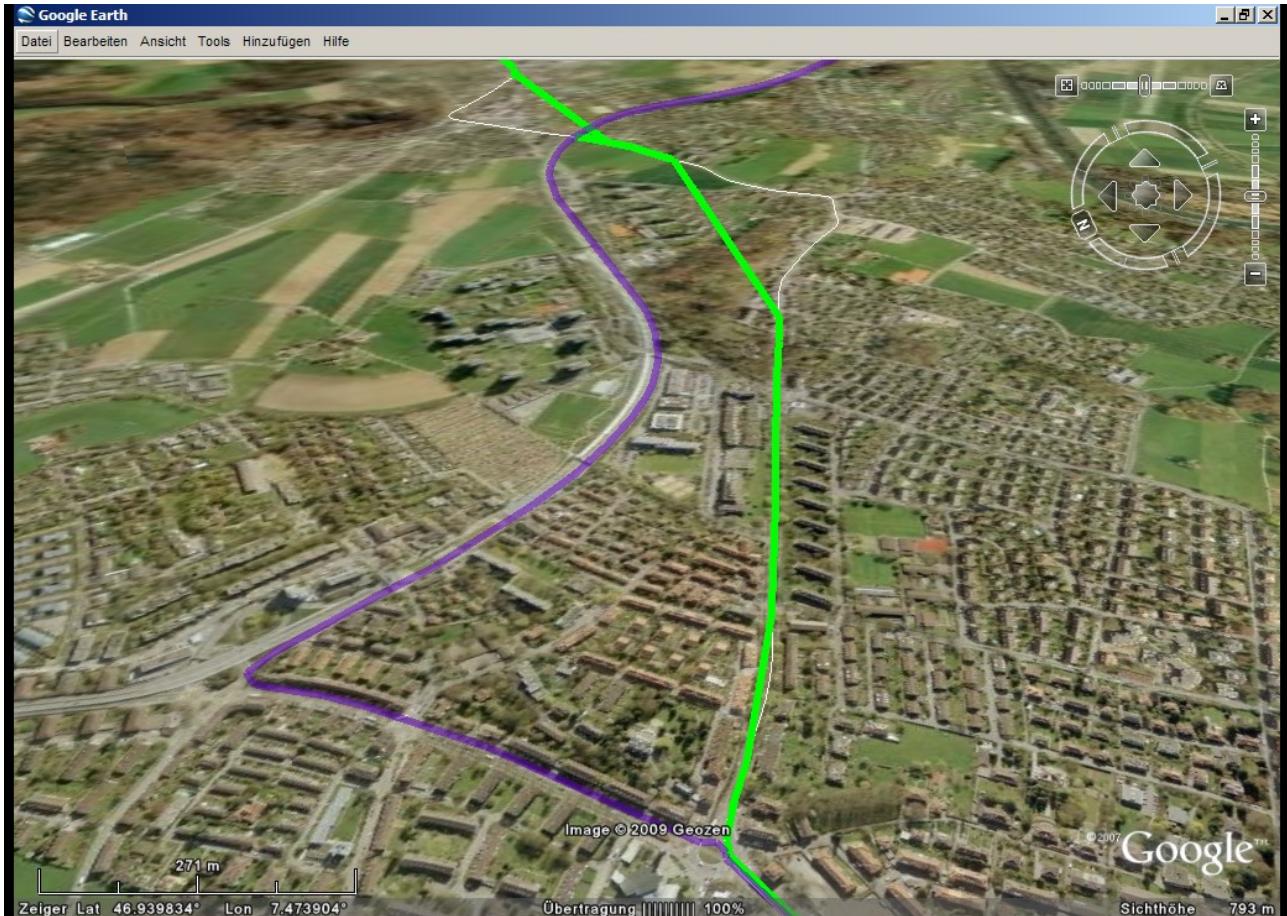


Illustration 43: At the Ostring

Implementation

When the distance to the track is large and the angle to track is also large, the use of only a single waypoint could lead to small irregularities, because of the aforementioned heading difficulties. This is visible at the position where the highway crosses the railroad track, and in extremity, when the car drives far away from the track to Spiez.

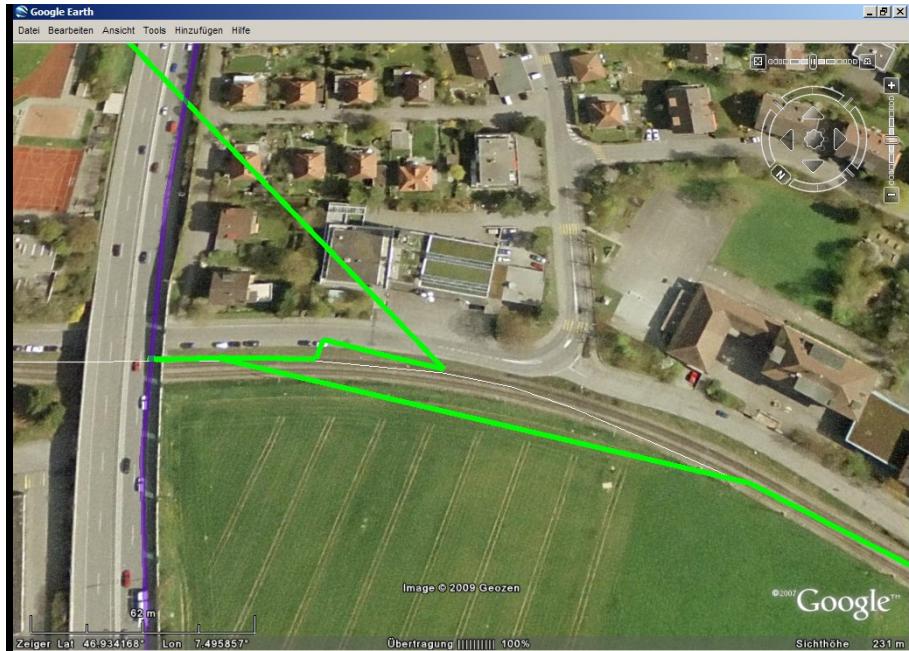


Illustration 44: In Gümligen

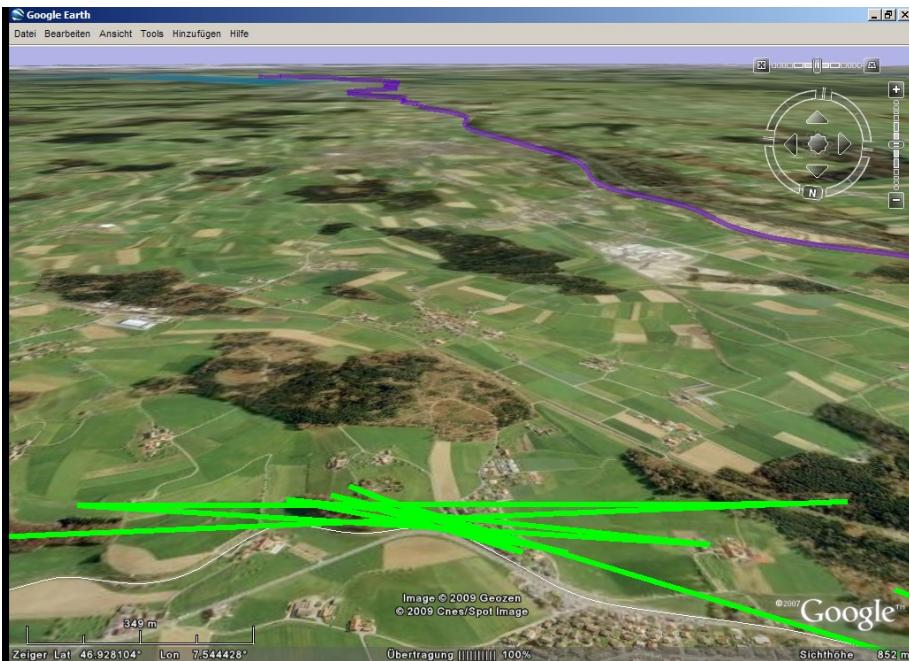


Illustration 45: Driving to Spiez

3.10 Class Estimate

The most important class in the whole solution is the Estimate class with it's interface IEstimate.

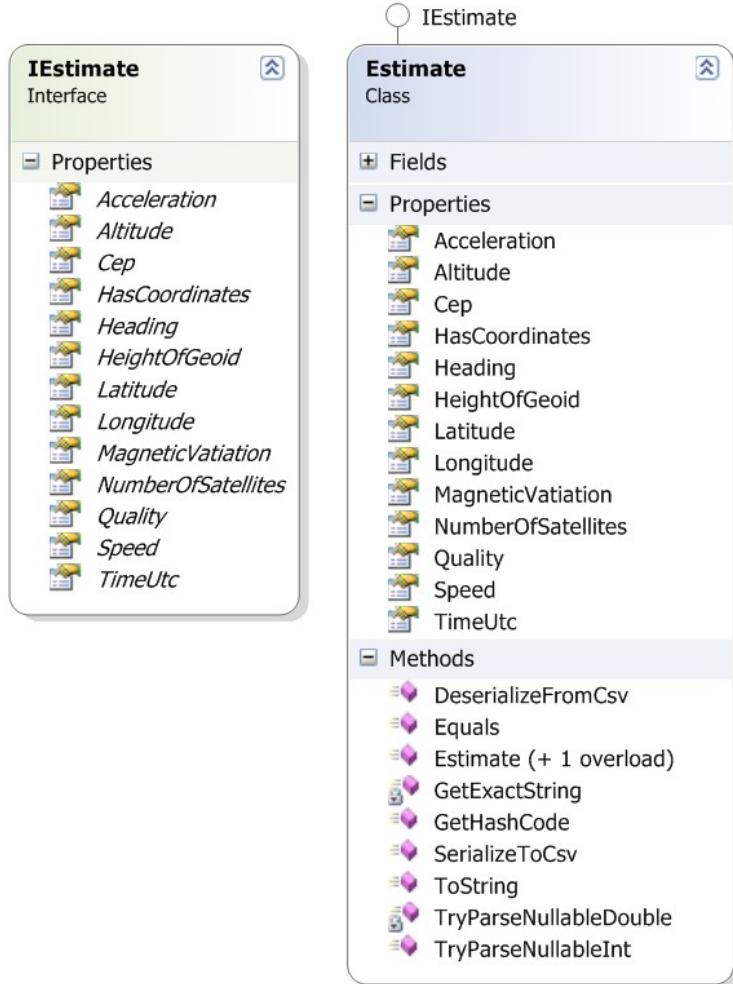


Illustration 46: The class Estimate with it's interface

Estimate objects are used to hold and distribute the positioning information from tool to tool in the chain. It contains all the data that is used throughout the chain path from the input to the output. It has serialization functionality, to CSV, which is used for logging.

The serialization to the NMEA Strings is in the emitter tool, [NmeaSerialEmitter](#).

3.11 Class DeviantDouble

The class DeviantDouble plays a central role in merging information. It consists of a double value, plus a standard deviation for that value.

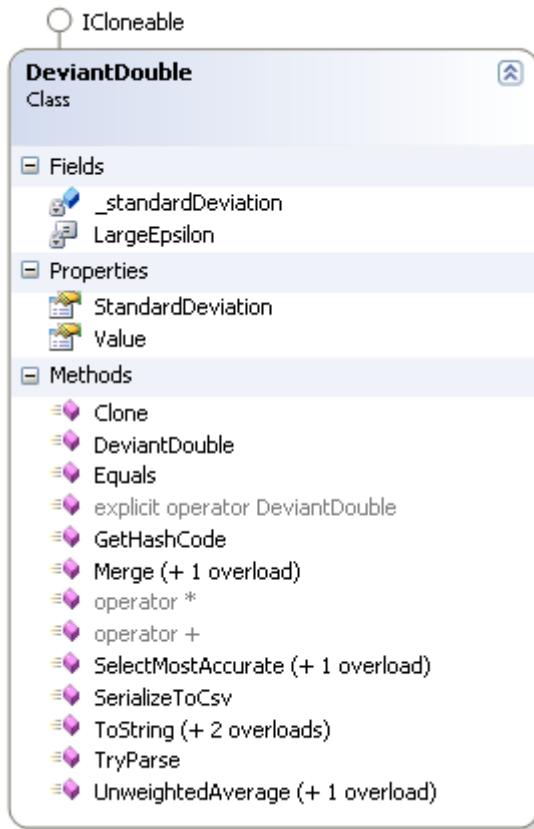


Illustration 47: Class DeviantDouble

The class has some specifically implemented operators that allow easy usage in the merger tool for certain calculations. The `Merge()` method merges two DeviantDouble objects into one, using their value and standard deviations to create a new DeviantDouble which represents the most probable value of the merged ones, having its own standard deviation.

The algorithm behind is a weighted average, as described in the analysis chapter.

3.12 Class Merger

The merger is the single most important tool. This is where all the estimates from the various input devices are blended (merged) into a single output.

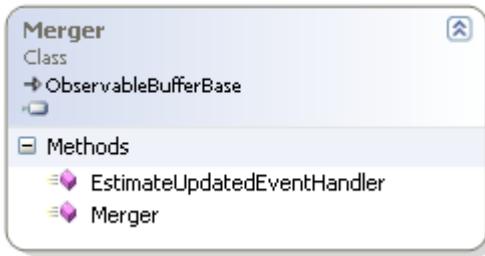


Illustration 48: Class Merger

The merging is done using an extrapolation from the last merged estimate to the current time and the current estimate. This slightly differs from the analysis where the estimates from all input sensors are individually extrapolated. I have found this unnecessary and only complicating the process.

3.13 Icons

I generally use the "Crystal Icons" by Everaldo Coelho.

New icons

I have created a new green "Play" Icon for better visualization of the working state of a tool. The original icon file was the player_play.png file and I have set the hue to -100, using GIMP. Currently the changed icon is only available as 16x16 pixel version and is called player_play_green.png

3.14 The GUI

The GUI is kept as simple as possible. It sports 3 areas, implemented as *Group Boxes*, that allow the input of the required data, and provide status information. Additionally, a *Property Grid* was used to show the current estimate in detail.

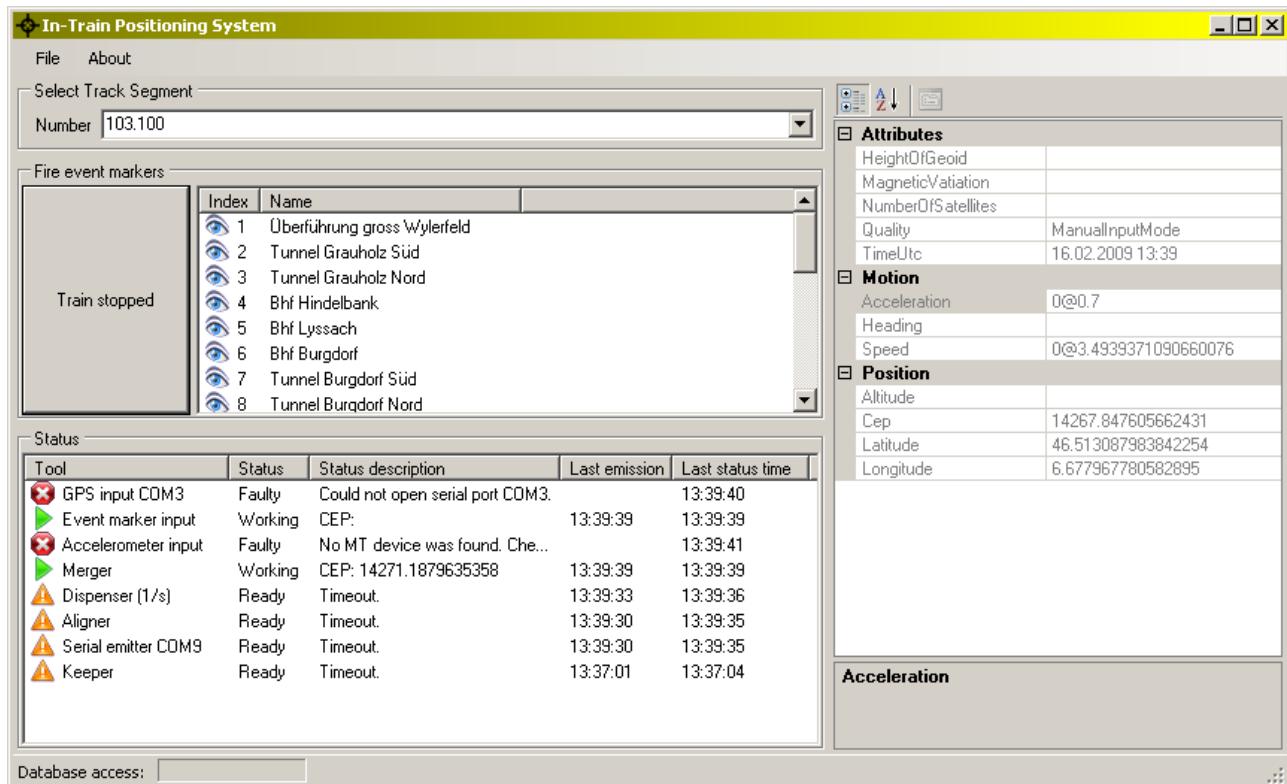


Illustration 49: The GUI of the system

3.14.1 Select Track Segment

Here, the user can select the current Track Segment using it's number. The number must be looked up in a table (available at the customer), if the correspondence between the track and it's number is not known to the user.

3.14.2 Fire Event Markers

Here, the user may fire the displayed event markers, which correspond to the previously selected track. Additionally, the user may tell the system when the train is completely stopped, via a dedicated button.

3.14.3 Status

On the status pane, the status of the various tools is displayed, according to the SRS.

- A green arrow means that the tool is working and has emitted an estimate not more than 3 seconds ago.
- An orange exclamation sign means, that the tools is ready to provide an estimate, but no data is available.

- A red cross sign means that the tool is in an error state and can not provide any information until the problem is solved.

3.14.4 Ergonomics

No specific care has been taken to have a good user experience in terms of GUI ergonomics. This should be added in a future version.

3.15 Tool chain

The tool chain is one of the central elements in the solution. It is implemented as own class, with interface, and provides a possibility to be derived from. Only one kind of tool chain is implemented the final solution. Other implementation of tool chains, especially for a kind of a "replay" feature would also be possible.

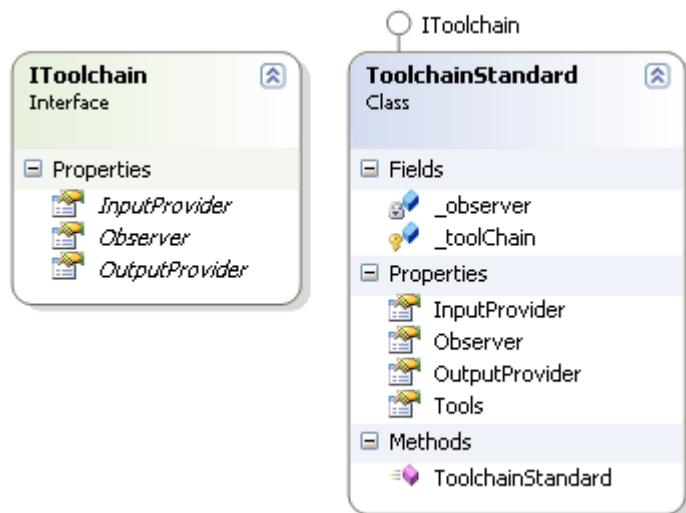


Illustration 50: The Toolchain class

The tool chain is built by registering the individual tools with each other. In the current class the **ToolchainStandard**, the tools are chained as follows:

```
//build chain
merger.Register(feeder);
merger.Register(mtidProvider);
merger.Register(gps);
merger.Register(keeper); //register on the keeper for input
keeper.Register(merger); //attach at output of merger as stub
dispenser.Register(merger);
if (Settings.Default.UseAligner)
{
    aligner.Register(dispenser);
    emitter.Register(aligner);
}
else
{
    emitter.Register(dispenser);
}
```

Illustration 36 shows, schematically, the implemented Tool chain with it's tools.

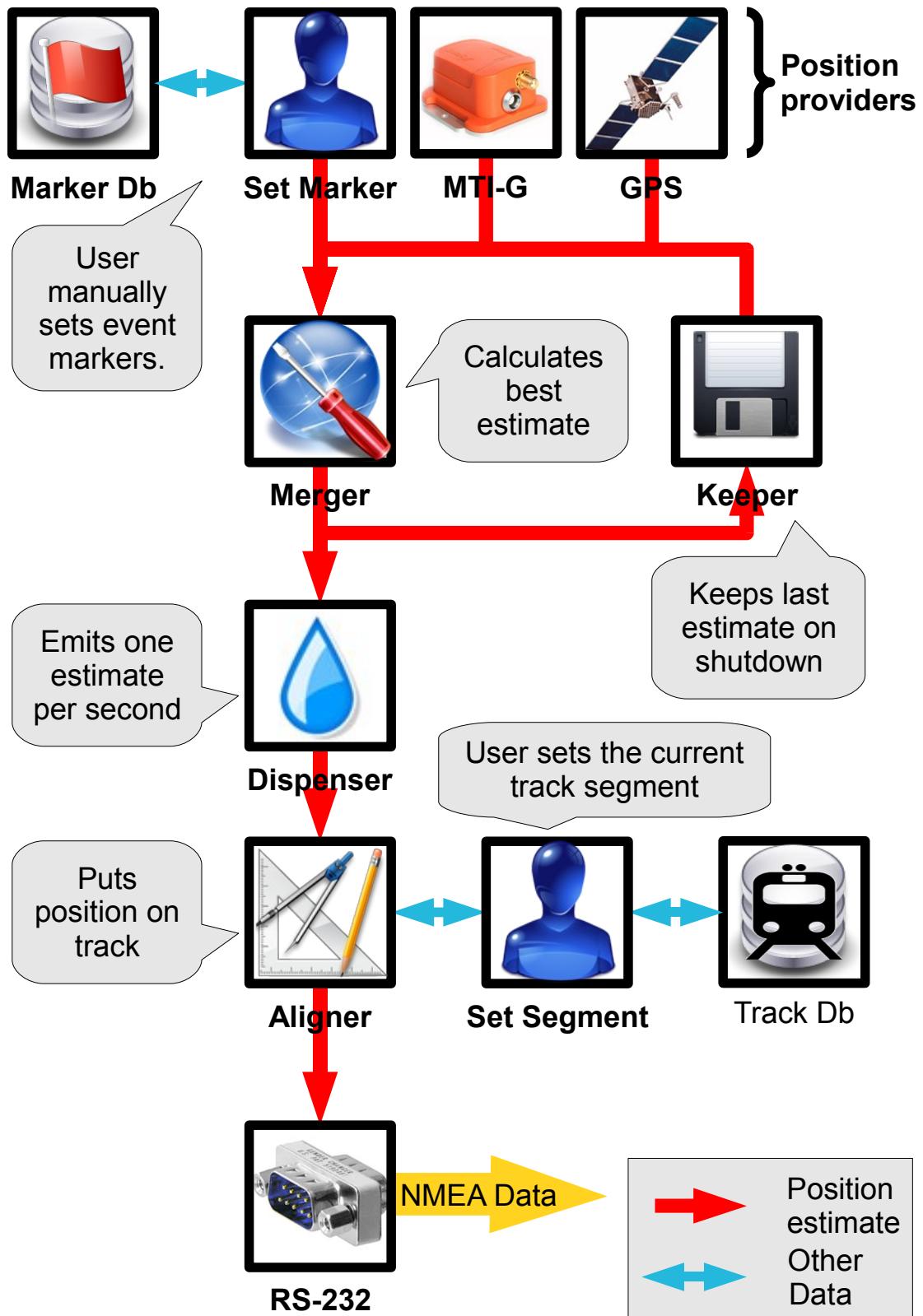


Illustration 51: Architecture overview of the implemented Toolchain

3.16 Configuration

The built-in mechanism of Visual Studio 2008 has been used to keep the application configurable. The main executable and some of the library projects have their own app.config files with a settings section in it. After installation on the target machine, the app.config file may get adapted to the situation on that machine.

Most of the settings are used to build the tool chain.

3.17 External Code and Libraries

Where possible and suitable external code of good quality has been used. Such code is attributed in the comments.

The largest external code block is the NMEA interpreter, which has been moved to an extra project. This code also has been slightly adapted.

For geodetic calculations, a geodesy library has been used. In the Common Package, some types are referenced, and thus used in the whole solution.

4 3rd Party component usage

4.1 Crystal Icons

These icons are used as status indicators in the GUI.

The crystal icon set from http://everaldo.com/crystal/crystal_project.tar.gz is from Everaldo Coelho. The used version was changes by yellowicon. The distributor kindly requests to include links to www.everaldo.com and www.yellowicon.com

4.1.1 License

LGPL

4.2 u-blox Evaluation Kit

The use of the u-blox GPS receiver requires the installation of a USB-driver. See <http://www.u-blox.com/>

4.2.1 License

The license grants all rights to use and install the software in combination with a u-blox hardware product.

4.3 NMEA Sentence Interpreter

This is a single class, written by Jon Person. See <http://www.geoframeworks.com/>

4.3.1 License

CPOL, <http://www.codeproject.com/info/cpol10.aspx>

4.4 C# Geodesy Library for GPS

This is a small library providing distance calculations on the surface of the earth. The library is available in source code at <http://www.gavaghan.org/blog/free-source-code/geodesy-library-vincentys-formula>

The library also defines coordinates and position classes which are used in the solution, and the types are now part of the common interfaces definition. These interfaces were simply generated from the signatures of the existing methods. The classes interfaced are Angle, GlobalCoordinates and GlobalPosition, with the interface classes IAngle, IGlobalCoordinates and IGlobalPosition respectively. The library has been adapted to properly implement those interfaces.

See <http://www.gavaghan.org/blog/>

4.4.1 License

There is no license attached to the code.

4.5 XSens SDK

The XSens SDK provides a COM DLL and documentation. See [XSNS08UM].

4.5.1 License

The SDK has a proprietary license, and we have only obtained a temporary license because the sensor is only rented.

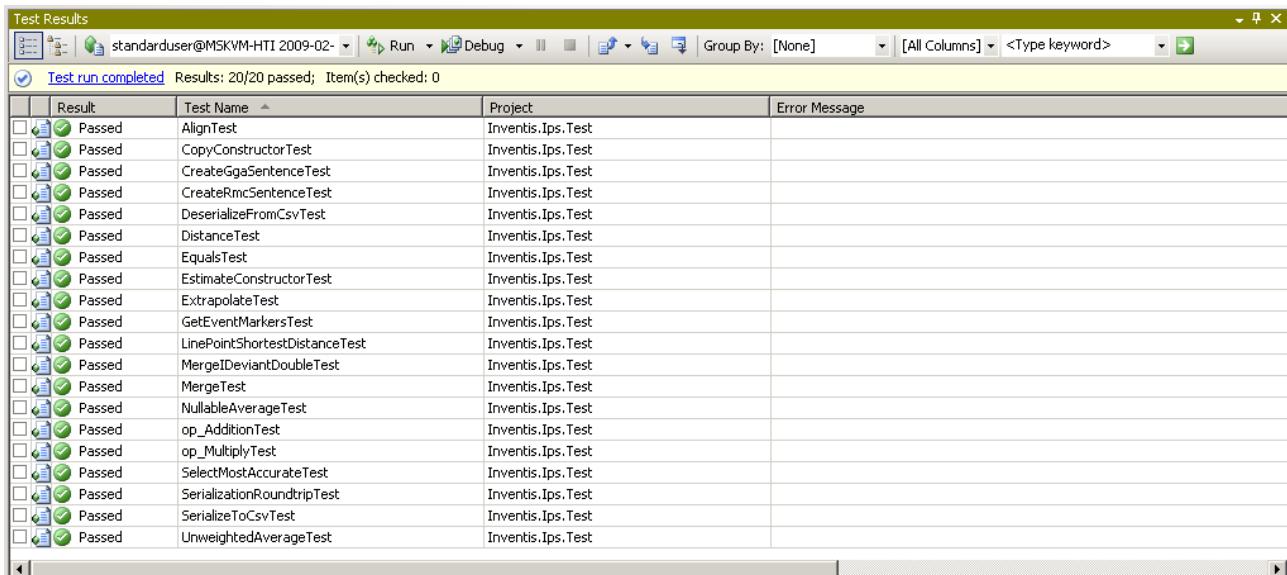
5 Test

5.1 Unit Tests

Unit tests are a means of testing parts of software systems.

I have created a single test project for the whole solution that tests critical parts of the solution.

Visual Studio 2008 supports a simple way of creating and executing Unit tests.



The screenshot shows the 'Test Results' window in Visual Studio 2008. The title bar says 'Test Results'. The status bar at the bottom indicates 'standarduser@MSKVM-HTI 2009-02-' and 'Test run completed'. The main area is a grid table with four columns: 'Result', 'Test Name', 'Project', and 'Error Message'. The 'Result' column contains green checkmarks for all 20 tests listed. The 'Test Name' column lists methods such as AlignTest, CopyConstructorTest, CreateGaSentenceTest, etc. The 'Project' column shows 'Inventis.Ips.Test' for all entries. The 'Error Message' column is empty.

Result	Test Name	Project	Error Message
Passed	AlignTest	Inventis.Ips.Test	
Passed	CopyConstructorTest	Inventis.Ips.Test	
Passed	CreateGaSentenceTest	Inventis.Ips.Test	
Passed	CreateRmcSentenceTest	Inventis.Ips.Test	
Passed	DeserializeFromCsvTest	Inventis.Ips.Test	
Passed	DistanceTest	Inventis.Ips.Test	
Passed	EqualsTest	Inventis.Ips.Test	
Passed	EstimateConstructorTest	Inventis.Ips.Test	
Passed	ExtrapolateTest	Inventis.Ips.Test	
Passed	GetEventMarkersTest	Inventis.Ips.Test	
Passed	LinePointShortestDistanceTest	Inventis.Ips.Test	
Passed	MergeIDeviantDoubleTest	Inventis.Ips.Test	
Passed	MergeTest	Inventis.Ips.Test	
Passed	NullableAverageTest	Inventis.Ips.Test	
Passed	op_AdditionTest	Inventis.Ips.Test	
Passed	op_MultiplyTest	Inventis.Ips.Test	
Passed	SelectMostAccurateTest	Inventis.Ips.Test	
Passed	SerializationRoundtripTest	Inventis.Ips.Test	
Passed	SerializeToCsvTest	Inventis.Ips.Test	
Passed	UnweightedAverageTest	Inventis.Ips.Test	

Illustration 52: Visualization of Unit Test test results in Visual Studio 2008

5.1.1 Application

Unit test are applicable everywhere, where a concrete method should get tested with concrete parameters. This also may help development, when the test is written beforehand. This is then called *Test-driven Development*¹¹. I have partially used this technique by writing test and code in parallel, especially when using the track database.

Unit tests have proven to be a very effective way of testing the outcome of small function blocks. It also enforces the thinking about the modularization of code.

5.1.2 Unit tests in the solution

During the course of the implementation, I have created 20 Unit tests, all of them are visible in the Illustration 54. The tests are mostly used to verify the various algorithms used.

Found errors

About a dozen otherwise quite hard to find errors have been found using the unit tests. An example of this is shown below.

Non-positive coordinate values

11 See http://en.wikipedia.org/wiki/Test-driven_development

There was a problem when using coordinates with non-positive values, as found east of Greenwich and south of the equator. A value estimated as absolute had a sign in the method "GetLongitudeStrings").

The correct line now is:

```
double lonMinutes = ((Math.Abs(estimate.Longitude.Value) - lonDegrees) * 60);
```

5.1.3 How to test events

Testing the raising of events may get easily tested via anonymous delegates. An example from <http://code-inside.de/blog/tag/unit-tests/>

shows how to do it:

```
[TestMethod]
public void ConnectionManager_Raise_StateChanged_Event()
{
    ConnectionManager man = new ConnectionManager();
    Assert.AreEqual(ConnectionStates.Disconnected, man.State);

    bool eventRaised = false;

    man.StateChanged += delegate(object sender, StateChangedEventArgs args)
    {
        eventRaised = true;
    };
    man.State = ConnectionStates.Connecting;

    Assert.IsTrue(eventRaised);
}
```

The test has a boolean "eventRaised". As soon as the event is raised, the anonymous delegate (the code right after the registration to the event), is called. The boolean value is changed in the delegate and this now can get asserted.

5.2 ***Test the fulfillment of the Requirements***

For each of the requirements in the SRS, the results of the test are shown below.

ID	Name	Pri o.	Result	Comment
1	NMEA Protocol	1	OK	EarthBridge can read the output.
2	NMEA Update	1	OK	EarthBridge shows these updates
3	NMEA serial configuration	1	OK	EarthBridge can read the output, as transferred.
4	NMEA speed	2	Not tested.	
5	RMC Sentence	1	Partly working.	Navigation receiver warning is not implemented Speed over Ground (in Knots!), is faulty, according to drive tests
6	GGA Sentence	3	Partly working.	Fix Quality indicator always invalid. Number of Satellites is not correctly reflected.

Test

7	GUI not mandatory	1	OK, but GUI must be started.	GUI is not separated, this is OK, due to Change Request.
56	GUI English	1	OK	
65	GUI translatable	1	OK	Solved by resource files.
8	GUI railway segment number	1	OK	
9	GUI segment as text	1	OK	
10	Track db input matching	1	OK	Wrong input is not possible, because of the list in the GUI is read-only.
11	Track db input plausibility	3	Missing	
12	Track db input success	1	OK	
13	Event firing	1	OK	
14	Stop event	1	OK	
15	Event list scrollable	1	OK	
16	Event list population	2	OK	
17	Event reselection	2	OK	
53	Event shortcut	2	OK, when the GUI has focus.	Space is used
54	Event double-click	1	OK	
18	Event re-firing	2	OK	

Test

19	Last event	2	Wrong	Last element stays selected.
20	Status at any time	2	OK	
21	Position Provider Status information	2	OK	
22	Output status information	1	OK	
23	Comport Adapter	1	OK, possible.	This not used in the project but possible with the respective hardware.
57	Virtual Comport	3	OK	
24	Engine start	1	OK	
25	GUI start	2	Requirement dismissed.	Dismissed by change request.
26	GUI stop	2	Requirement dismissed.	Dismissed by change request.
27	Engine stop	1	OK	
28	WiFi scan	2	Missing	
29	WiFi lookup	2	Missing	
30	GPS usage	1	OK	
31	IMU usage	1	OK	
32	Track db usage	1	OK	
33	Track as waypoints	1	OK	Also, a heading information is used, because the existing database has this information available.
34	Unique segment number	1	OK	
35	Segment narrowing	1	OK	

Test

36	Using event markers	1	OK	
37	Log output	1	OK	
38	Log disabling	2	Missing	
39	Log file	2	OK	
41	Blending inputs	1	OK	
42	Position estimation	1	OK	
43	Estimation CEP	3	Not reached	CEP grows very fast over time, about 500m per minute, when no GPS is available.
44	Waypoint count	1	OK	
45	AP count	2	Not tested	
46	Maximum speed with GPS	1	Not tested	
64	Maximum speed with any sensor	3	Not tested	
47	Db update	2	OK	Possible by replacing whole database.
48	Using laptop	1	OK	
49	Using 3 rd party libs	1	OK	No non-commercial components are used, except the virtual comports. This component is easily replaceable.
55	Engine separated	2	Requirement dismissed.	Dismissed by change request.
63	Emulation mode	1	Partly implemented	Not changeable by configuration.
50	NMEA output	1	OK	Works with <i>EarthBridge</i> .
51	Coding Guidelines	1	OK	
52	Availability	2	Not tested	

58	Quick Start Guide	1	OK	
59	Documentation	1	OK	
60	GPS Accuracy	2	Not tested	
61	WiFi Accuracy	2	Not tested	WiFi is not implemented.
62	Inertial Accuracy	2	OK	Done while driving in a bus.

5.2.1 Summary

Of all requirements with priority 1, the following have not been fully met:

5, 63

Of all requirements with priority 1, the following have not been tested:

46

5.3 **Test drives**

To evaluate the function of the IPS, some test drives in the S-Bahn around Berne were done.

For these test drives, no database was used, thus not using the user-clicked event markers and the alignment features. The reason for this is, that the CEP behavior of the system is better visible in this configuration.

5.3.1 Drives

Several drives were taken, two of them described in this section. The sensors were installed near the window, and the track was saved using built-in logs and with the *EarthBridge* tool.

Test

Test



Illustration 53: Test drive equipment

Gümligen-Belp

The train was boarded in Gümligen, with a GPS fix. The fix was occasionally lost during difficult landscapes, but was always regained. The GPS was placed inside a jacket pocket, and not, as would have been optimally, near the window. The drive was in a train from Gümligen to Rubigen, then in a bus from Rubigen to Belp.

The drive showed, that the heading is very wrong when no GPS is available this resulted in great slopes. Also the acceleration is measured with a slight negative bias. This resulted in in a quite great displacement of about 500 meters after about one minute.

Belp-Berne

The train was boarded in Belp, without a GPS fix. After a short time, a fix as available and the position was known. The fix remained stable during most of the journey or was regained quickly.

The track is visible below:

Test

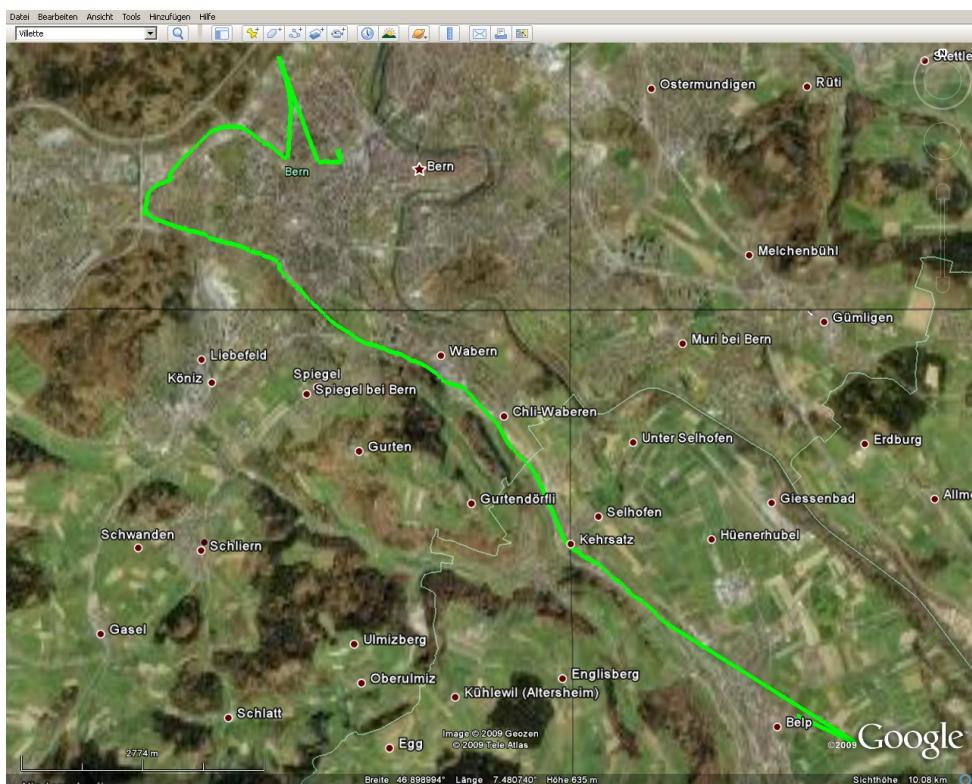


Illustration 54: Recorded track between Belp and Berne

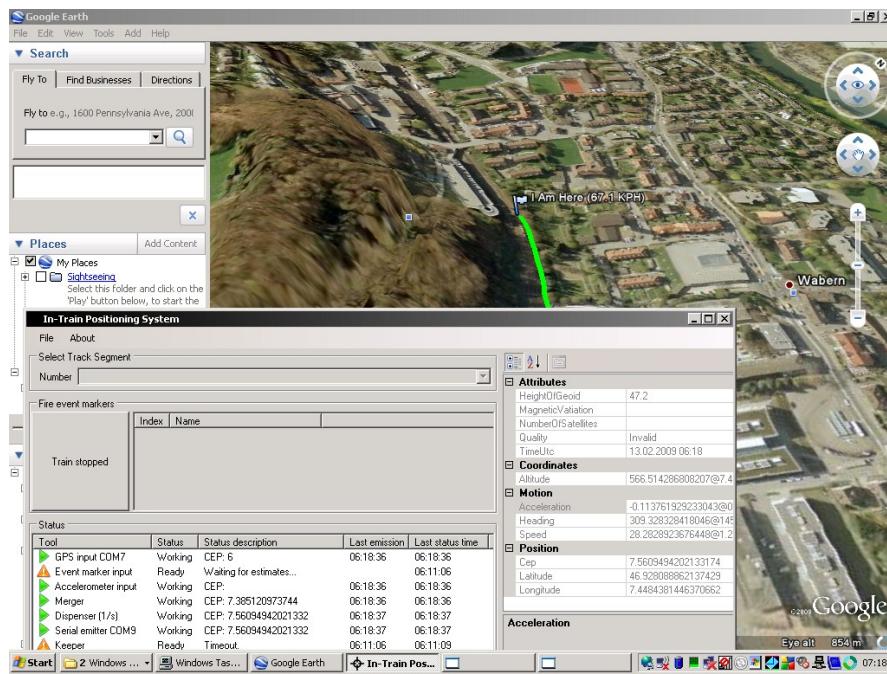


Illustration 55: Live position tracking using Earth Bridge

The positioning was quite good as the observation of the landscape suggested. Problems occurred when no GPS fix was available. The positioning deviated quite fast in these situations. This happened on three occasions, once in a tunnel at the Station Ausserholligen, once in a tunnel near the Berne main station, and after the train entered the Berne Main Station.

Test

At Ausserholligen, the deviation was not severe, as the fix loss only lasted short. The Screen shot below shows the problem as the train enters the tunnel below the building.

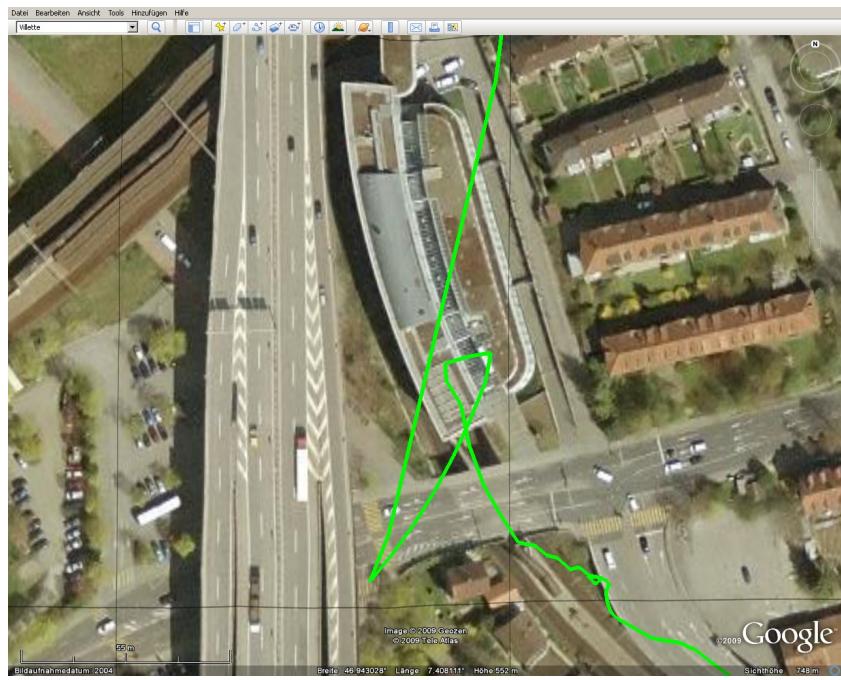


Illustration 56: Deviation starts at entering the tunnel below the building

At the tunnel near the Main Station, the deviation also starts at entering the tunnel. It is clearly visible, that there is a heading mismatch, as the estimated moving continues laterally. Also, since the slope evolved faster than the train moved and also grew much larger than the distance of the tunnel actually is, there seems to be a problem at the speed estimation or speed merging. This should get investigated further.

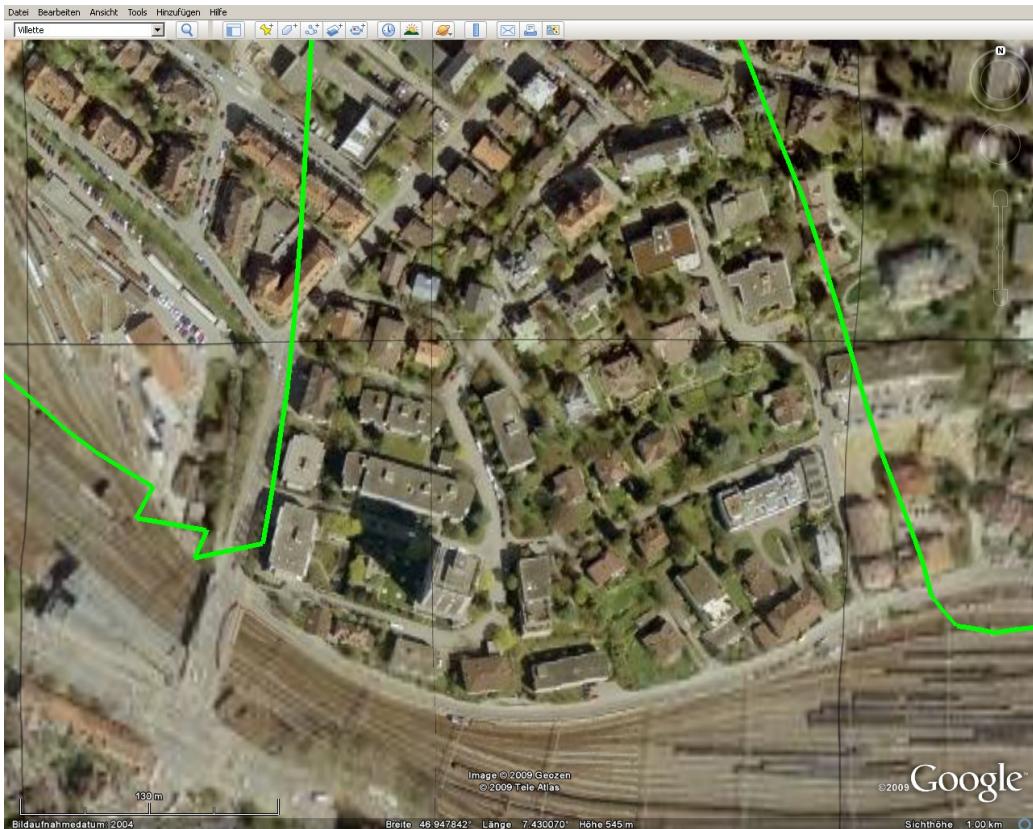


Illustration 57: Deviation at the tunnel near Bern Main Station

At the Bern Main Station, the same problem as above is visible. The deviation was not so fast, but still clearly visible. On the Google Earth data, the "Welle" a building over the position where the train stopped is missing.



Illustration 58: Position estimate derivation at the Bern Main Station

5.3.2 Detected problems and behavior

Most of the time, as expected, GPS is available. During these times, the positioning is quite accurate and will be aligned very well by the alignment feature.

If no GPS is available, the estimated position quite largely differs from the real position as observing the landscape from within the moving train. Once, the heading obtained from the MTi-G is inaccurate most of the time. This has already been observed during the development in the office, but the reason for this is still unclear. Second, there seems to be a speed mismatch between the integrated speed and the one obtained from the GPS. This is probably a bug in the conversion of the units. This may be the reason for the large "Slopes" observed during GPS outages.

5.3.3 Conclusion

The software works stable, but the output, in case of no GPS available may be much to be improved, probably with moderate effort. The main problems is the heading and the speed calculation, together with the bad quality of the acceleration measurements.

6 Project Management

6.1 Review Meetings

6.1.1 Kick-Off, 9.10.2008

On the Kick-Off, the following agreements have been made:

- The student delivers a schedule together with the SRS until 14.11.2008
- The next project review meeting takes place on the 11.11.2008
- The student discusses and defines the confidentiality level with the company.

6.1.2 Meeting on 11.11.2008

On the meeting the NDA was signed, the SRS reviewed and the time schedule discussed.

Open tasks

- A new system diagram is drawn by the student with the system boundaries tied to the software only.
- The student updates the SRS and the time schedule accordingly and sends both to the expert.

6.1.3 Meeting on 16.12.2008

In the Meeting, the current work, mainly Analysis, Synthesis and Design was discussed. No specific action was agreed upon, since the project seems to be on track.

Next Meeting will be at 13th of January 2008, 8:00am.

6.1.4 Meeting on 13.1.2009

In the Meeting, the documentation, the time schedule and the currently implemented code was discussed. No specific action was agreed upon. The project is slightly under pressure, since major parts are not yet implemented.

The next review will be at the 11th of February.

The examination of the thesis will be at 3rd of March 2009.

6.1.5 Meeting on 11.2.2009

The code has been partly reviewed and the project status has been evaluated.

The following points were raised during the discussion:

- The track data and event marker data seems to miss the railway stations at the beginning and the end of the track segment.
- The result of the evaluation of the DeviantDouble equality at the NaN value is unclear. This could be verified with a Unit test.
- The interface library may depend on compiled external code. It is not necessary to introduce an interface for types out of the Gavghan library. These interface could get removed.
- DeviantDouble could implement an explicit cast from double, instead of a constructor.

Project Management

- DeviantDouble could have it's own Formatter using a format like "[N6@N8](#)" or similar.
- Implement an own TryParser for DeviantDouble

A change request was raised to omit the WCF part of the implementation. There is no specific benefit for the customer if WCF were used, but time is saved.

Deliverables

At the end date, only the documentation is to be delivered. The code will be reviewed together at the examination meeting.

7 Known Issues

7.1 Currently known Issues

Currently the following issues are known and should get fixed in a later release:

- The end points of tracks (the starting and terminating railway station), when a track segment is selected in the GUI, is not shown and also not considered for the alignment feature. In the example below, the Station "Casinoplatz" should appear as first entry in the list.

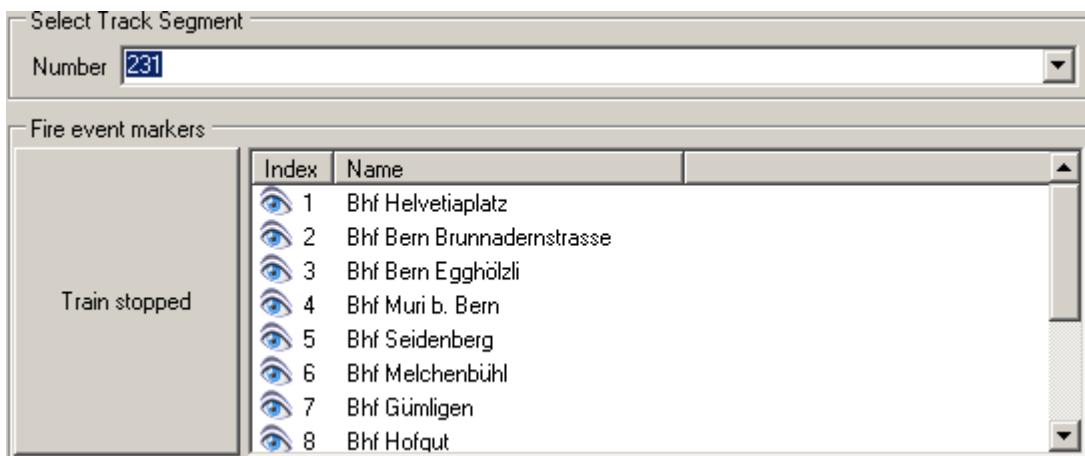


Illustration 59: Missing starting railway station for a selected track segment

- The heading measurements of the MTi-G are often very inaccurate. This results in wrong driving direction during GPS outages.
- The acceleration measurements are inaccurate. This is by the limited accuracy of the sensor hardware itself, but there are also possibilities for a better implementation.
- The database is more often accessed as probably necessary. This slows down the system. The proxy class should get improved.
- The install script is missing. Installation of IPS is not done automatically.

7.2 Further improvements in the code

First, the known issues should get addressed.

Further, the following improvements could be made:

- Improve the integration algorithm by using more accurate timing than the one provided by the DateTime Class. This will result in more precise numerical results.
- When merging, also extrapolate the current value, in case its time stamp is not of the current time. This is not considered in the current solution.
- Better evaluate the heading deviation for GPS and the MTi-G sensor. This will result in better merged heading information.
- In the aligner tool, also merge the heading into the estimate and calculate a new CEP based on the distance to the track
- The settings could be converted from the currently used app.config in a separate class and made editable within the application itself.

Known Issues

- In case the GPS is not sending data, an error should be shown. Currently only "Waiting for Estimates" is shown.
- Improve the GUI ergonomics. Especially design the input navigation using a keyboard.

8 Lessons learned

May lessons have been learned. Among the most important:

- Be always more conservative with the time schedule than you think.
- Search the Internet and ask colleagues – most probably, someone has already done what you want to do.
- Visual Studio 2008 is great
- Make prototypes.

9 Future

Although the thesis work has been finished, the project still could evolve further. Some of the possible future work and improvements is shown in this chapter.

9.1 Optimizations

9.1.1 GUI

The GUI could get ergonomically improved and more option for user data input could be thought of, especially an improved handling of the "Train stopped" button, where the button could also explicitly remain pressed for a certain amount of time.

9.1.2 AHRS and Kalman

The merging of the estimates could be done with advanced mathematical algorithm, as the *Kalman Filter* and *Bayesian Estimators*, often used in commercial *AHRS* systems.

9.1.3 GPS

With GPS, slow changes when the train is completely stopped, could get averaged out.

9.1.4 Improve Acceleration measurements

The acceleration measurements could be done in true 3D, to be independent of the sensor orientation.

Additionally, the measured acceleration could get calibrated using the GPS when available. The bias of the accelerometer measurements is quite high.

9.2 Extensions to the sensors

9.2.1 Sensors (Position providers)

In addition to the position providers used in this thesis, others are also possible.

Optical stop recognition

Using a low cost camera and image processing, a complete stop of a train could be detected by the number of pixels in a sequence of images which are changing their color. Above a certain ratio, the train is considered as stopped.

Optical landmark recognition

Using a fast camera, artificial or natural landmarks could be detected and compared to a geo-referenced database of the landmarks. A library for artificial landmarks is available: ARToolkit, <http://www.hitl.washington.edu/artoolkit/>

This technology has been used for an indoor positioning system for an Augmented Reality project: <http://studierstube.icg.tu-graz.ac.at/projects/mobile/SignPost/>

Optical speed measurements

Using two cameras (or two line cameras) and the known distance between the two, the traveling speed would be deductible from the time used to travel the distance between the cameras.

RFID

RFID's located along the tracks could be detected and looked up against a database with georeferenced entries of the RFID's. It would have to be determined whether passive ones are detectable fast enough and over the required distance.

10 Bibliography

[NICO01]	Nicomsoft, Advanced WiFi-Manager, http://www.nicomsoft.com/wifiman/advanced.htm
[NETS01]	Marius Milner, stumbler dot net, http://www.stumbler.net/
[PLLB05] (currently not used)	Anthony LaMarca, Yatin Chawathe, Sunny Consolvo, Jeffrey Hightower, Ian Smith, James Scott, Tim Sohn, James Howard, Jeff Hughes, Fred Potter, Jason Tabert, Pauline Powledge, Gaetano Borriello, Bill Schilit: Place Lab: Device Positioning Using Radio Beacons in the Wild,
[SUM08TE]	Themeneingabe Master Thesis Thema In-Train-Navigation, 21.8.2008, Marcel Suter. Obtainable from the author.
[SUM08FS]	In-Train Positioning System MAS-06-02.23 Feasibility Study, 10.10.2008, Marcel Suter. Obtainable from the author.
[IEEE98]	IEEE Recommended Practice for Software Requirements Specification, 1998, The Institute of Electrical and Electronics Engineers, Inc. 345 East 47th Street, New York, NY 10017-2394, USA, ISBN 0-7381-0448-5, SS94654 (PDF)
[SUM08SR]	In-Train Positioning System MAS-06-02.23 Software Requirmement Specification, 17.11.2008, Marcel Suter. Obtainable from the author.
[GPSI08]	Dale DePriest, NMEA data, 2008, http://www.gpsinformation.org/dale/nmea.htm
[XSNS08UM]	Xsens Technologies B.V.: MTi-G User Manual and Technical Documentation, Available from Xsens Technologies B.V, Pantheon 6a, P.O. Box 559, 7500 AN Enschede, The Netherlands, www.xsens.com , 2008
[TOPC08IB]	Lbackstrom: Geometry Concepts: Basic Concepts, Available at http://www.topcoder.com/tc? module=Static&d1=tutorials&d2=geometry1#line_point_distance
[PLST08AE]	Peter Lamb, Sylvie Thiébaux: Avoiding Explicit Map-Matching in Vehicle Location, CSIRO Mathematical and Information Sciences, GPO Box 664, Canberra ACT 2601, Australia, available at http://users.rsise.anu.edu.au/~thiebaux/papers/its99.doc

11 Thanks

This thesis would not have been possible in the presented way without the help of many people.

I would like to thank:

- My wife, Sara, for her patience during this time
- The expert, Rolf Wenger for sharing his knowledge in a kind and comprehensive way
- The supervisor, Stefan Bigler, for helping me out of dead ends
- Michael Birchmeier, of ENKOM Inventis, for the project idea and for spending the amount of money necessary
- The guys at Redmond, for building the great environment I was able to use for this piece of software.

Appendix A: Mindmap of the analysis

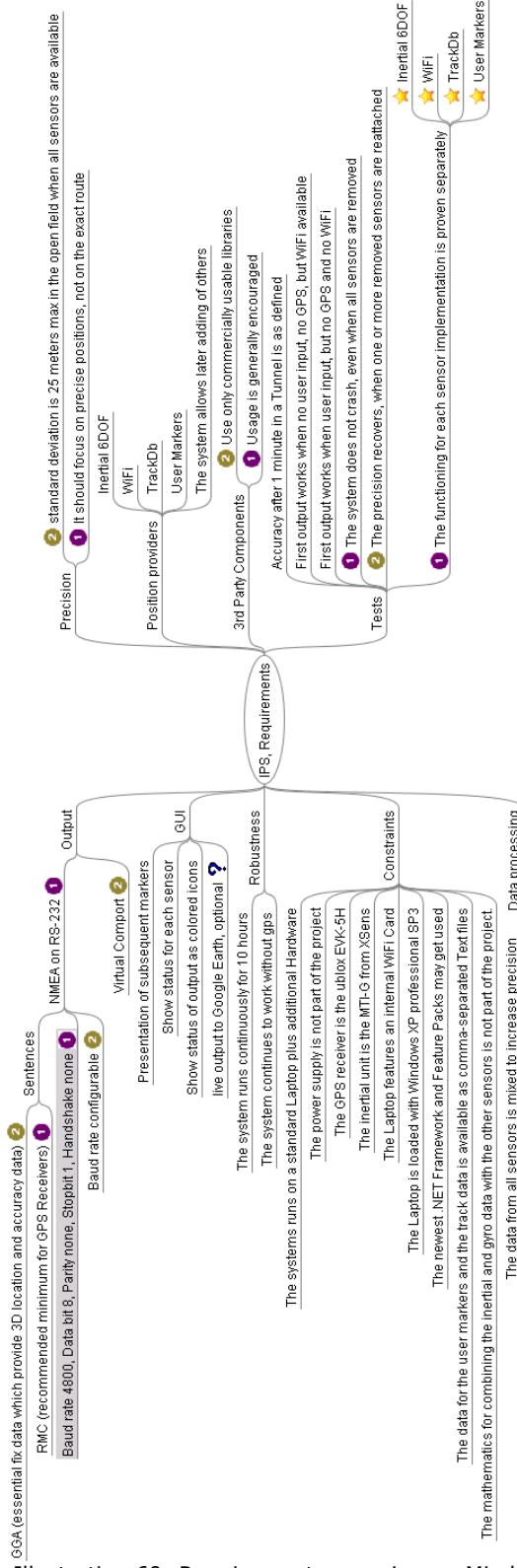
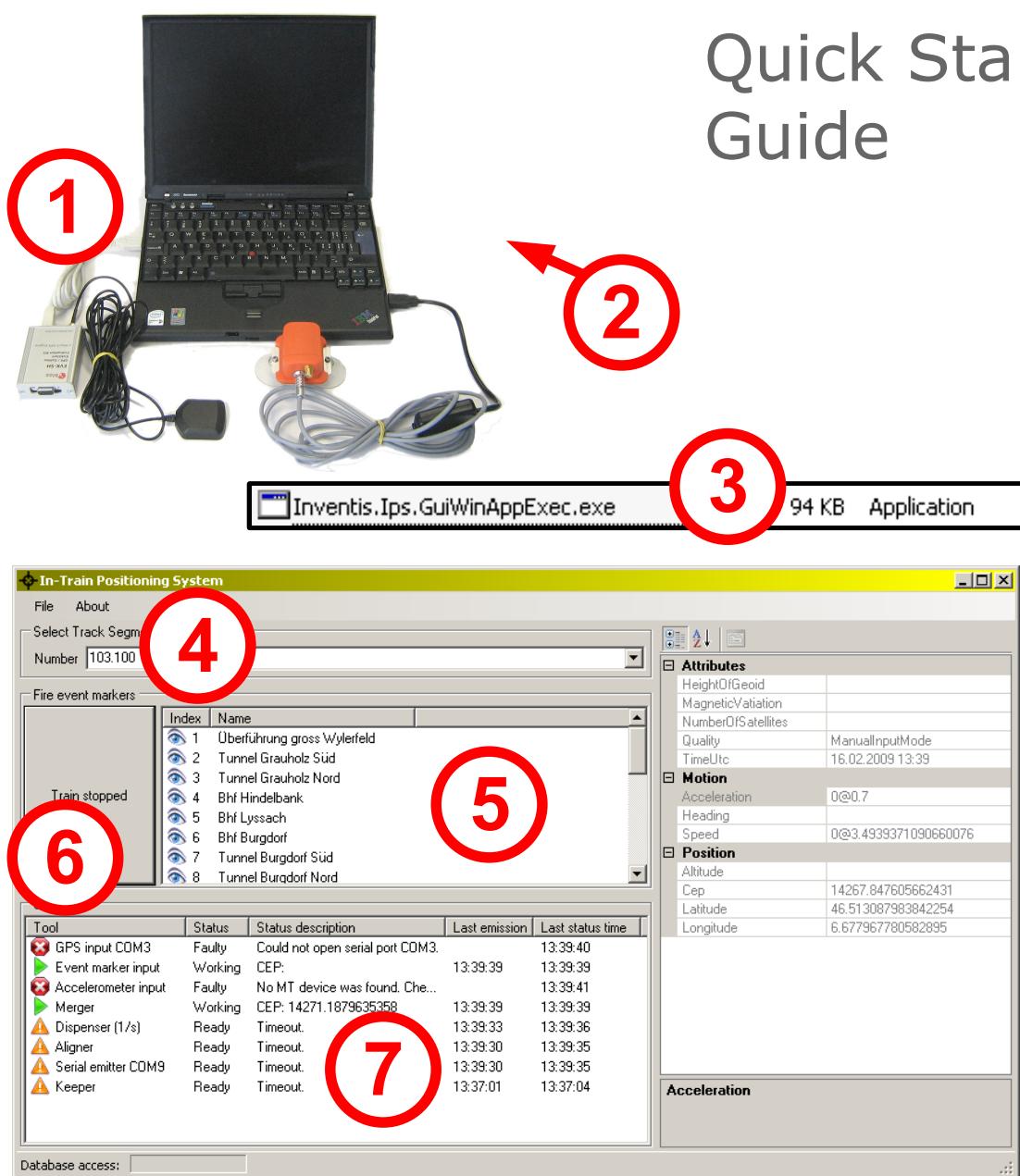


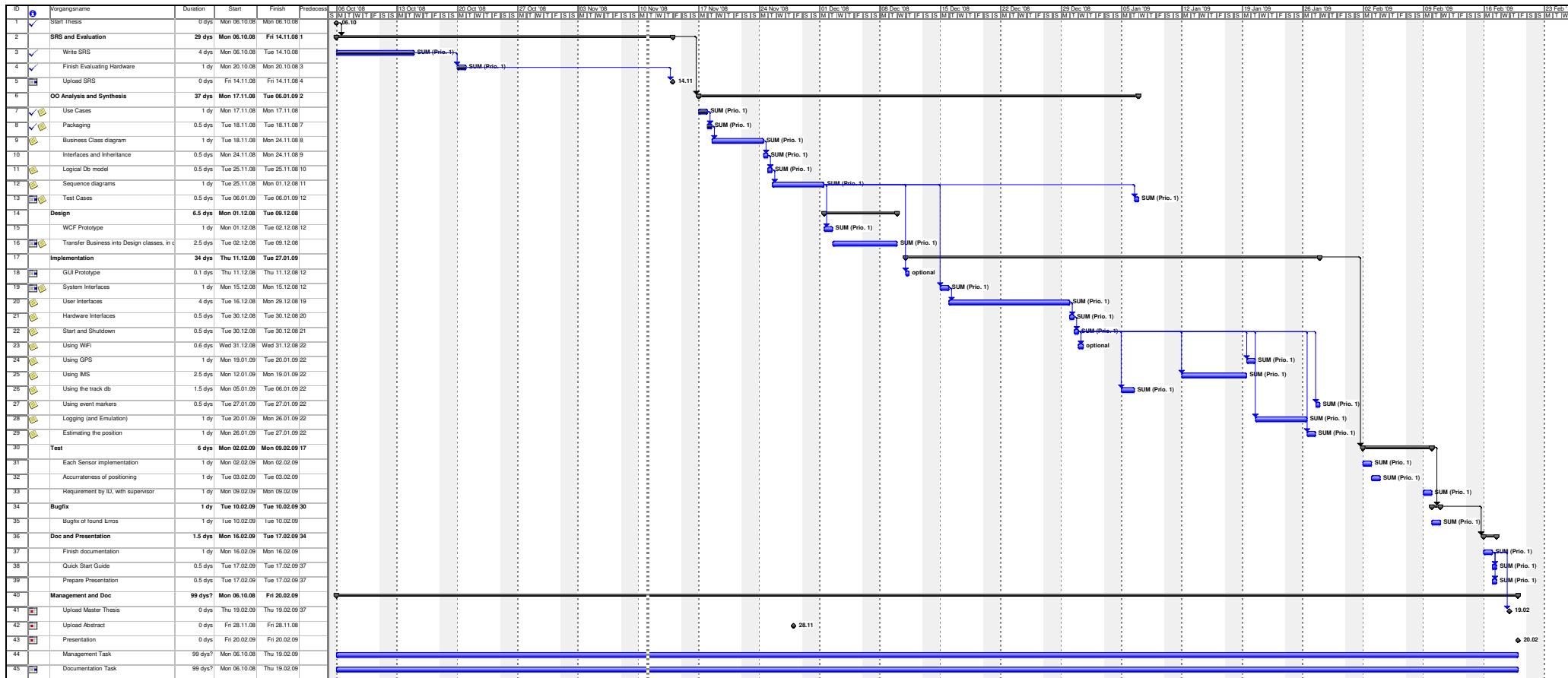
Illustration 60: Requirements overview as Mind map

In-Train Positioning System

Quick Start Guide



- 1. Attach the external GPS and Acceleration Sensor to the Laptop**
- 2. Attach the NMEA-Consuming device to the serial port of the Laptop**
- 3. Start the Application from Windows Explorer by double-clicking on the "Inventis.Ips.GuiWinAppExec.exe" file.**
- 4. Select the Track Segment by choosing the number from the drop-down list.**
- 5. Double-click on the event markers when appropriate**
- 6. Single-click on the "Train stopped" button when appropriate**
- 7. Watch the status information**



7	Use Cases Identify and model use cases. Define scenarios and test the use cases	
8	Packaging Create packages, package diagram.)GUI and Engine must be strictly separated. Also all code that is not part of this diploma work, be it publicly available open source or from the customer, goes into separate packages()	
9	Business Class diagram Create business classes, Class diagram	
10	Logical db model Create logical database model	
12	Sequence diagrams Create sequence diagram for a position estimate from each position provider, Create activity diagram if necessary	
13	Test Cases Specify test cases and test the model with the scenarios	
16	Transfer Business into Design classes, in code	
	Model Business classes into Design classes, apply OO design patterns, Design associations, attributes, methods, enumerations	0.5
	Design the interfaces, Design the packaging	0.5
	Design the database tables	0.5
	Study .NET Unit testing and define test cases (if enough time available)	1
19	System Interfaces Find suitable NMEA lib, Implement e-reader, copy data from current estimate into the NMEA classes	1
20	User Interfaces	
	Languages, Designing the main parts of the dialog	0.5
	Design the WCF classes for Event firing and track segment number	
	input	1
	repopulate event markers	0.5
	Event firing	1
	Implement status providing and GUI Elements, WCF for this.	1
21	Hardware Interfaces Install adapter, test output with a prototype	0.5
22	Start and Shutdown	
	Create starter EXE, probably a Windows service for the engine, Create Starter EXE for the GUI	0.5
23	Using WiFi Test Placelab	1
	Get Data	1
	Integrate Placelab	1
24	Using GPS Integrate GPS	1
25	Using IMS	
	Get acquainted with the sensor and the data access	1
	Create position estimates with correct CEP from the data.	0.5
	Design and implement a simple algorithm that links the last known position with the IMS output. (Some sort of back-calculating the acceleration and turnrate error)	1
26	Using the track db	
	Create track db proxy or list that hold current track waypoints (Probably narrowed down by user input). Calculate nearest point algorithm	1.5
27	Using event markers Create corresponding position provider	0.5
28	Logging (and Emulation)	
	Create logger, reading from logs, emulating sensor	1
29	Estimating the position Create blender	1